WYDAWNICTWO SIGMA NOT

Cena 12 000 zł

P. 1877/91

Biblioteka Śląska

4 1991

informatyka

Kompilatory firmy Jensen & Partners International już dostępne w Polsce!

TopSpeed Modula-2 (programowanie obiektowe...)
TopSpeed C 100% ANSI, programowanie współbieżne...)

Produkty firmy JPI są od dwóch lat światową rewelacją. Bezkonkurencyjna jakość generowanego kodu, w pełni zintegrowane środowisko ze znakomitym edytorem i wszelkimi narzędziami potrzebnymi profesjonalnemu programiście – to czynniki, które złożyły się na końcowy sukces tej firmy.

Wszystkie kompilatory – w przygotowaniu: TopSpeed Pascal, TopSpeed C + + (wiosna) i TopSpeed Ada – pracują w tym samym środowisku, co umożliwia realizację dużych projektów przy użyciu różnych języków i bez konieczności poznawania wielu, różniących się od siebie narzędzi. Dzięki temu można kompilować programy napisane na Turbo Pascalu firmy Borland.

Zintegrowane środowisko kompilatorów JPI to m.in.:

* edytor (9 okien, zaznaczanie bloków kolumnami, tablica ASCI, kalkulator, możliwość definiowania klawiszy, ...),
* pomoc w formie Hyper Textu,
* SmartLinker wybierający z modułów tylko używane procedury,
* Visual Interaktiv Debugger pozwalający na uruchamianie i śledzenie programów (także współbieżnych) – interfejs do programów uruchomieniowych innych firm,
* debugger „post-mortem" (!)
* biblioteka z pełnym kodem źródłowym,
* grafika własna lub możliwość korzystania z bibliotek firmy Borland
* TopSpeed: Assembler, Profiler, Disassembler,
* PROM Locator (sterowniki przemysłowe),
* i wiele innych użytecznych narzędzi dla profesjonalnych programistów.

Wyżej wymienione oprogramowanie oferujemy w wersjach: profesjonalnej (DOS), profesjonalnej rozszerzonej (DOS) oraz profesjonalnej rozszerzonej (OS/2). Otrzymacie Państwo również licencję na sprzedaż programów napisanych przy użyciu zakupionych kompilatorów.

Proszę do nas dzwonić lub pisać w celu otrzymania dodatkowych informacji. Na życzenie wysyłamy pełną specyfikację przedstawionych pakietów. Oferujemy również dodatkowe biblioteki tzw. toolkits JPI oraz możliwość sprawdzenia oprogramowania na indywidualne zamówienie.

MERCURIA LTD
ul. Powstańców Śląskich 95
53-332 Wrocław

telefon (071) 60 56 79
telefaks (071) 61 67 11
teleks (071) 24 91 opg pl

0/4/91

informatyka

nr 4 1991
Kwiecień
Rok wydania XXVI

Miesięcznik
ISSN 0542-9951
INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
dr inż. Wacław ISZKOWSKI
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
dr inż. Wojciech MOKRZYCKI
mgr inż. Jan RYŻKO
mgr Hanna WŁODARSKA

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA:
Wydawnictw Czasopism i Książek
Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych
redakcja nie zwraca

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji lub Działu Reklamy i Marketingu
00-950 Warszawa
ul. Biała 4
telefon: 20-31-24
telefaks: 20-31-16
teleks: 814550

W numerze:

	Strona
Systemy CAI – *Jan Piecha*	1
Systematyczne testowanie programów – zalety i wady (1) – *Dorota Kuchta*	8
Programowanie obiektowe – *Leszek Nowaczyk*	12
Od sieci neuronowych do neurokomputerów – *Tomasz Berus, Bohdan Macukow*	14
/unix	
Brzydka Zielona Książka – *Jacek Surma*	19
Jak wpływać na proces (w shellu Bourne'a) – *Jan Walasek*	21
zBITki	
COLOR Gra – *Wacław Iszkowski*	III okł.

W najbliższych numerach:

- **Andrzej Górecki omawia komputeryzację handlu detalicznego przy użyciu kas systemowych, wyposażonych w laserowe czytniki kodów kreskowych.**
- **Kamil Skudlarski charakteryzuje Analizator Uogólnionych Stochastycznych Sieci Petriego.**
- **Cezary Dołęga opisuje problemy i metody wyszukiwania znaków w rozpoznawaniu tekstu drukowanego**
- **Jacek Surma i Jan Walasek kontynuują cykl materiałów na temat UNIX-a.**

Warunki prenumeraty

Przyjęcie prenumeraty – wyłącznie na podstawie dokonanej wpłaty na drukach dostarczanych dotychczasowym prenumeratorom przez Wydawnictwo, lub nowym – po uprzednim zgłoszeniu zapotrzebowania (pisemnie lub telefonicznie) w Zakładzie Kolportażu Wydawnictwa.

Blankiet wpłaty – powinien zawierać następujące informacje: dokładna nazwa i adres (z kodem pocztowym) zamawiającego, tytuły zamawianych czasopism, ich liczbę i okres prenumeraty.

Wpłata – zgodnie z podanymi cenami należy dokonać w banku lub w UPT na konto podane na naszym blankiecie, tj:
Państwowy Bank Kredytowy III O/Warszawa nr: 370015-1573-139-11

Prenumeratorzy zbiorowi – osoby prawne obowiązują blankiety „Wpłata-Zamówienie". Cena normalna.

Prenumeratorzy indywidualni – osoby fizyczne obowiązują blankiety typu przekazy dla wpłat na rachunki bankowe. Cena normalna.

Prenumerata ulgowa – zgodnie z podaną ceną ulgową przysługuje wyłącznie osobom fizycznym, będącym członkami SNT, studentom i uczniom szkół zawodowych. Uczniowie szkół ogólnokształcących mogą zamówić w prenumeracie ulgowej tylko miesięcznik „Aura"
Uwaga! w podanym okresie prenumeraty można zamówić tylko po jednym egzemplarzu z każdego tytułu.

Prenumerata ze zleceniem wysyłki za granicę – cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa od ceny normalnej.
Należy podać dokładny adres odbiorcy za granicą.

Terminy przyjmowania prenumeraty:
– do 10 listopada na I, II, III, IV kwartał następnego roku
– do 28 lutego na II, III, IV kwartał br.
– do 31 maja na III i IV kwartał br.
– do 31 sierpnia na IV kwartał br.
Zmiany w prenumeracie, np. zmiana liczby tytułów, liczby egzemplarzy, rezygnacja z prenumeraty, można zgłaszać tylko w podanych terminach z mocą obowiązującą od następnego kwartału.

Egzemplarze archiwalne (z lat ubiegłych)
Można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa, ul. Mazowiecka 12 (tel. 26-80-16) lub zamówić pisemnie w Zakładzie Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek lub za zaliczeniem pocztowym.

Informacji o prenumeracie udziela: Zakład Kolportażu Wydawnictwa SIGMA-NOT Spółka z o.o., 00-716 Warszawa, ul. Bartycka 20, skr. 1004. Telefony: 40-00-21 wewn. 293, 295, 299 lub 40-30-86, 40-35-89.

Wstępna cena jednego egzemplarza na 1991 rok: normalna – 12 000 zł ulgowa – 8400 zł

Wartość prenumeraty (w zł):
Normalna: kwartalna – 36 000, półroczna 72 000, roczna 144 000
Ulgowa: kwartalna – 25 200, półroczna 50 400, roczna 100 800

Uwaga: W przypadku zmiany cen w okresie objętym prenumeratą, prenumeratorzy zobowiązani są do dopłaty różnicy cen.

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 48/91 n. 3650 egz.

Artykuł zawiera zbiór wniosków z pracy badawczej realizowanej w latach 1986–1990 w Uniwersytecie Śląskim, poświęconej przydatności minikomputera w procesie dydaktycznym. Praca była finansowana z centralnego programu badawczego CPBP 08. 17.II D, koordynowanego przez Instytut Badań Pedagogicznych i Instytut Kształcenia Zawodowego w Warszawie.

JAN PIECHA
Zakład Elektroniki i Informatyki
Uniwersytet Śląski

POLITECHNIKA ŚLĄSKA Biblioteka Główna w Gliwicach
P. 1877/91

Systemy CAI

Nieustanny przyrost wiedzy prowadzi do ciągłej redukcji czasu przeznaczonego na nauczanie określonych partii materiału. Efektem tego jest:
- ujednolicenie programów nauczania, bez zwracania uwagi na indywidualne zainteresowania,
- uśrednienie czasu realizacji tematów dla wszystkich uczniów (studentów), niezależnie od indywidualnych zdolności,
- mała aktywność ucznia (studenta) w trakcie realizacji zadań dydaktycznych, z uwagi na duże tempo pracy nauczyciela,
- słaba i wyrywkowa kontrola postępów w opanowaniu wiedzy.

Fakty te można uznać za podstawową sprzeczność procesu dydaktycznego.

Jednym ze sposobów wykonywania pracy dydaktycznej jest tzw. *programowe nauczanie*. Realizacja wybranego materiału odbywa się na podstawie wielowariantowego przewodnika umożliwiającego dokonanie indywidualnego wyboru drogi przejścia przez zadany program nauczania. W tym modelu nauczania jest możliwa daleko idąca indywidualizacja nauczania pod względem treści i czasu realizacji zadań dydaktycznych.

Indywidualizacja nauczania stwarza poważne problemy związane z warsztatem dydaktycznym, tj. wyposażeniem pracowni i kadrą nauczycielską. Realizacja takiego zadania nie jest możliwa bez automatyzacji procesu przekazywania wiedzy.

W kryteriach projektu systemu programowanego nauczania należy uwzględnić następujące cztery zasady:
1) wyodrębnienia ściśle ze sobą powiązanych partii nauczanego materiału,
2) odpytywania po każdej partii materiału i natychmiastowej oceny odpowiedzi,
3) indywidualizacji tempa i treści nauczania,
4) realizacji lekcji na podstawie materiałów poddanych wielostopniowej weryfikacji empirycznej.

Zarządzanie systemem programowanego nauczania jest realizowane na podstawie drugiej zasady, pod warunkiem pomyślnej realizacji zasady pierwszej. Zasada czwarta wymaga od autora materiałów do programowanego nauczania przystosowania stopnia trudności materiałów do średnich możliwości percepcyjnych ucznia (studenta) danego kursu.

Automatyzacją procesu dydaktycznego są zainteresowane obie strony, tzn. nauczający i uczący się. Zainteresowanie nauczyciela wynika z faktu, że chce uczynić swoje zajęcia bardziej atrakcyjne, bogatsze w treści merytoryczne, wreszcie bardziej efektywne. Zainteresowanie ucznia (studenta) wynika z chęci poznania nowych metod i nowych narzędzi dydaktycznych.

W dziedzinie teorii programowanego nauczania opublikowano wiele prac, jednak przypadków wdrożeń jest niewiele lub są fragmentaryczne. Wynika to głównie z trudności związanych z opracowywaniem odpowiedniego podręcznika. Ponieważ komputer został skonstruowany z myślą o algorytmicznym i wielowariantowym rozwiązywaniu problemów, kwestia opracowania materiałów do programowanego nauczania wspomaganego komputerem znacznie się upraszcza.

Modele dydaktyczne

Pracę nad sposobem nauczania należy rozpocząć od zdefiniowania modelu lekcji. Podjęte przed laty prace na temat programowanego nauczania posłużyły za punkt wyjścia dla autorów pierwszych lekcji wspomaganych komputerem. Próby usystematyzowania opracowań oparto na pewnych ogólnych kryteriach wynikających ze znanych modeli dydaktycznych, stosowanych w systemach programowanego nauczania. Do najbardziej rozpowszechnionych można zaliczyć cztery modele dydaktyczne: liniowy model Skinnera, testy Presseya, strategia Crowdera i strategia Kaya.

W *liniowym modelu Skinnera* każda porcja informacji jest zamykana kilkoma pytaniami. Krok następny podejmuje się niezależnie od wyniku sprawdzianu. Co najwyżej są odnotowywane częściowe wyniki składające się na końcową ocenę. Liniowa realizacja lekcji ma wiele wad deklasujących tę metodę. Najbardziej nużące dla zdolnego ucznia (studenta) jest powtarzanie niewielkich partii materiału w celu uzyskania pozytywnych odpowiedzi w całej grupie. Taka procedura postępowania nie wpływa pozytywnie na efektywność nauczania mierzoną dla całej grupy.

W *testach* opracowanych przez *Presseya* założono wybór jednej z kilku możliwych odpowiedzi (tylko jedna jest prawidłowa). Metoda zaproponowana przez Presseya była wstępem do lekcji, której treść była formułowana na podstawie zbioru pytań i odpowiedzi.

Strategia jest złożonym algorytmem, w którym są analizowane różne warianty rozwiązania przed podjęciem ostatecznej decyzji, podobnie jak w przypadku gier inteligentnych: szachów, brydża itp. W *strategii Crowdera* proponuje się realizację części materiału na zasadach zgodnych z liniowym modelem Skinnera. Po przeprowadzonym teście przewidziano kilka rozwidleń programów, zależnie od wyników tego testu. Komputer wybiera drogę kontynuacji programu stosownie do postępów ćwiczącego. W przypadku odpowiedzi błędnych następuje powrót do punktu wyjściowego z możliwością wprowadzenia dodatkowego materiału wyjaśniającego.

W *strategii Keya* zakłada się kilka poziomów wiedzy w programie CAI. Błąd popełniony w trakcie realizacji programu na k-tym (trudniejszym) poziomie powoduje automatyczne przełączenie stopnia trudności na poziom k-1 – bardziej szczegółowy (łatwiejszy). I odwrotnie, dobre postępy w trakcie realizacji lekcji powodują stopniowe podnoszenie stopnia trudności programu CAI.

W porównaniu z programami liniowymi programy rozgałęzione w zdecydowany sposób zwiększają efektywność nauczania, umożliwiają indywidualny dobór treści i tempa pracy.

Koncepcja CAI

Systemy wspomagania nauczania mikrokomputerem są znane w literaturze pod nazwą CAI (ang. *Computer Assisted Instruction*), CBT (ang. *Computer Based Teaching*) lub CAE (ang. *Computer Aided Education*). Pierwsze badania nad zastosowaniem komputera w procesie dydaktycznym podjęto prawie równocześnie z innymi zastosowaniami, tzn. przed kilkudziesięciu laty. Z uwagi na szczególnie duży koszt sprzętu oprogramowania, praktyczne zastosowanie komputera w procesie dydaktycznym nie miało szans realizacji.

Rozwój techniki mikrokomputerowej, zwłaszcza znaczna obniżka cen sprzętu oraz rozwój narzędzi programowania, spowodowały wzrost zainteresowania oprogramowaniem użytkowym, również w sferze dydaktycznej.

CECHY CAI I METODY WARTOŚCIOWANIA ODPOWIEDZI

W trakcie opracowywania systemów CAI należy mieć na uwadze kilka modelowych cech obiektu projektowania, jak:
– orientacja treści,
– takt realizacji CAI,

– format obrazu,
– forma pytań,
– rozwidlenia lekcji,
– techniki pomocnicze,
– organizacja pamięci zewnętrznej.

● **Orientacja treści**

Do najważniejszych cech systemu CAI należy właściwa orientacja treści. Chodzi o to, aby program nie pozostawiał wątpliwości w sprawie realizacji następnego kroku. Jeśli w zaproponowanej formie programu rytmiczny przebieg lekcji jest uzależniony od dodatkowych komentarzy ze strony nauczyciela, to oznacza, że materiał CAI (lub jego fragment) został źle opracowany.

Taką wadę należy uznać za podstawową przyczynę następujących skutków:
– straty czasu w procesie nauczania,
– negatywnego stosunku uczącego się do wartości programu,
– utraty motywacji dla tej formy uczenia się.
Uwagi te można odnieść do wielu programów opracowanych przez dobrych programistów, którzy lekceważą sprawę wartości metodycznych lekcji. Brak cech orientacji programu ma być, zdaniem autora takiego programu, kompensowany faktem obecności instruktora na lekcji. Jest to niestety założenie błędne, w zasadniczy sposób zmniejszające wartość programu.

Przez właściwą orientację programu należy również rozumieć możliwość powrotu do fragmentu lekcji, która została niewłaściwie zrozumiana lub zapomniana. Zadaniem projektanta CAI jest zapewnienie możliwości prostego wyjścia z każdej sytuacji, bez uwag krytycznych pod adresem uczącego się.

● **Takt realizacji**

Jedną z podstawowych technik tworzenia materiału CAI jest zasada ekspozycji informacji na ekranie w postaci kolejnych warstw; krok po kroku, poczynając od informacji elementarnych dla danego obrazu, kończąc na treściach w pełni objaśniających zagadnienie. Częstym błędem popełnianym przez autorów systemów CAI jest narzucanie tempa realizacji całej lekcji lub jej fragmentów. Stosowana technika automatycznego odmierzania czasu jest jedną z najbardziej deprymujących form oceny uczącego się.

Automatyczne odmierzanie czasu oznacza:
– niemożność dopasowania taktu do tempa realizacji lekcji przez każdego ćwiczącego,
– pozbawienie uczącego się podstawowych możliwości sterowania przebiegiem lekcji,
– sztuczność i niekomunikatywność systemu CAI.
Godna uwagi jest możliwość stosowania powtarzalnych form graficznych w celu skierowania uwagi uczącego się na nowe elementy wiedzy, bez potrzeby rozpoznawania elementów wcześniej poznanych. Jednak i w tym przypadku takt realizacji lekcji nie może być automatyczny. Przejście do kolejnej warstwy informacji powinno odbywać się w takt uczącego się (przez naciśnięcie wybranego klawisza).

● **Format obrazu**

Nadmiar informacji eksponowanej na jednym obrazie staje się przyczyną utrudnionej komunikacji z użytkownikiem CAI. Obowiązuje generalna zasada budowania obrazów zawierających niezbędne minimum informacji w końcowej ekspozycji ekranu. Zadaniem projektanta systemu CAI jest dokonanie takiego podziału treści lekcji, aby uczynić zadość tej zasadzie. Niektóre narzędzia wspomagające opracowanie programów dla CAI są wyposażone w odpowiednie mechanizmy ograniczające wymiar zbioru danych przekazywanego na ekran. Właściwy dobór proporcji tekstu do form graficznych jest bardzo ważnym elementem projektu CAI. Tam gdzie to jest możliwe, należy tekst ograniczać do niezbędnego minimum.

Przyjmując takie założenie stawiamy sobie pytanie, czy można ten warunek spełnić niezależnie od przedmiotu CAI. Odpowiedź brzmi NIE, a drastycznym tego przykładem może być program wspomagający nauczanie języka. Można jednak postawić drugie pytanie. Czy komputerowe wspomaganie nauczania jest jednakowo efektywne dla każdego przedmiotu? Z pewnością nie i można na potwierdzenie tej tezy znaleźć wiele programów, które w opinii wytrawnego metodyka urągają podstawowym zasadom dobrego nauczania. Co więcej, uwagi krytyczne nie wynikają z błędów merytorycznych lub błędów ze sfery sztuki informatycznej, lecz z faktu, że obiekt CAI jest niepodatny na tego rodzaju opracowanie.

● **Forma pytań**

Pytania należą do kluczowych problemów systemów CAI. Dlatego technika stawiania pytań, ich ocena oraz formy komentarza w odpowiedziach wymagają szczególnej uwagi w projekcie CAI. Umiejętnie sformułowane pytanie może być podstawą wyboru dalszej drogi realizacji lekcji. Poniżej przedstawiono pięć podstawowych metod oceny odpowiedzi stawianych w trakcie realizacji lekcji wspomaganej komputerem.

Oczekiwana odpowiedź dobra

Lekcja CAI może zawierać pytania kwitowane odpowiedzią, którą porównuje się z wzorcową odpowiedzią prawidłową. Jest to najprostsza metoda oceny odpowiedzi, najczęściej stosowana w systemach CAI realizowanych za pomocą małych komputerów osobistych.

Najistotniejszą wadą tej metody jest brak możliwości wyselekcjonowania odpowiedzi dobrej, nieco inaczej sformułowanej niż w podanym wzorcu. W tym przypadku największą trudność dla autora systemu CAI stanowi konieczność takiego sformułowania pytań, aby liczba możliwych odpowiedzi osiągnęła minimum. Następnie należy wprowadzić reguły formułowania odpowiedzi (np. zdefiniować słownik odpowiedzi) tak, aby była możliwa jednoznaczna odpowiedź.

Jeśli przedstawione zabiegi nie dadzą oczekiwanych rezultatów, to należy opracować listę wzorcowych odpowiedzi na każde pytanie. Rozwidlenie programu dla tej klasy pytań dotyczy jedynie komentarzy mających na celu naprowadzenie uczącego się na właściwą odpowiedź.

Oczekiwana odpowiedź zła

Porównywanie uzyskanej odpowiedzi ze zbiorem możliwych odpowiedzi złych prowadzi do bardzo złożonego modelu ocen. Pomijając odpowiedzi bezsensowne, można zwykle wyodrębnić kilka grup odpowiedzi prawdopodobnych (w tym również odpowiedzi dobre). Przyporządkowanie uzyskanej odpowiedzi do odpowiedniej grupy oznacza określenie stopnia zrozumienia (lub niezrozumienia) omawianego tematu. W ten sposób następuje rozwidlenie lekcji, wywołanie komentarza i ponowny powrót do fazy pytań. Operacja powtórzeń może być wykonywana nieograniczoną liczbę razy lub w *pętli liczonej*, jeśli zechcemy ograniczyć liczbę błędnych repetycji.

Do tej techniki oceny można zaliczyć metodę **wyznaczania odpowiedzi**. Gdy odpowiedź jest bliska pozytywnej, komputer dokonuje jej korekty i podaje komentarz popełnionych błędów lub niejednoznaczności.

Odpowiedź nieoczekiwana

Bez względu na to, jak wiele wiemy na temat obiektu lekcji, czy jak dobry jest program CAI, niektórzy uczniowie (studenci) udzielają nieoczekiwanie złych odpowiedzi. W takim przypadku nie ma możliwości przyporządkowania odpowiedzi do jakiejkolwiek klasy. Jedynym wyjściem z tej sytuacji jest komentarz *odpowiedź nie rozpoznana* lub temu podobny.

Brak odpowiedzi

Przy opracowywaniu algorytmu oceny odpowiedzi należy również przewidzieć taką możliwość, że w trakcie realizacji lekcji ćwiczący nie jest w stanie udzielić żadnej rozsądnej odpowiedzi na postawione pytanie. Po upływie określonego czasu lub po naciśnięciu wybranego klawisza lekcja powinna być kontynuowana. W zasadzie najwłaściwszym wyjściem jest powrót do początku lekcji z wywołaniem dodatkowego komentarza.

Pomiar opóźnienia odpowiedzi

Jedną z ważnych zalet komputera jest możliwość automatycznego odmierzania czasu odpowiedzi. Jak się okazuje, przekroczenie zadanego limitu czasu kończy się z reguły odpowiedzią złą. Zbyt długi czas oczekiwania staje się również przyczyną utraty rytmu lekcji. Najlepszym rozwiązaniem w takim przypadku jest przerwanie stanu oczekiwania na odpowiedź i zaproponowanie pomocy ćwiczącemu, np. *czy chcesz dodatkowych wyjaśnień? T/N*. Oczywiście tego rodzaju powroty powinny zostać odnotowane w tablicy ocen (błędów) ćwiczącego.

Po zakończeniu serii pytań należy podać listę popełnionych błędów z odpowiednim ich komentarzem, zalecając powtórzenie określonych partii materiału tej lekcji lub partii poprzednich. Jest to bardzo istotna cecha CAI, jeśli przyjmiemy założenie, że są one wykorzystywane do pracy indywidualnej ucznia (studenta) w trakcie repetytorium wiedzy.

● **Rozwidlenia lekcji**

U podstaw realizacji strategii są rozwidlenia dróg przebiegu lekcji. Są one rezultatem analizy udzielanych odpowiedzi. Podstawą realizacji rozwidleń programów CAI są pytania sklasyfikowane powyżej.

W wyniku rozwidleń programu są realizowane następujące zadania systemu CAI:
– powtórki serii pytań,
– powtórki wybranych części lekcji,
– wywołanie dodatkowych komentarzy,
– przełączenie poziomu szczegółowości lekcji.

PRACA INDYWIDUALNA

Indywidualizacja pracy dydaktycznej dotyczy następujących przypadków:
– pracy w grupie, ale z indywidualnym tempem dostosowanym do umiejętności uczącego się,
– pracy w grupie według indywidualnego programu kształcenia,
– pracy indywidualnej z samokształceniem,
– samodzielnego repetytorium przerobionego materiału dydaktycznego.

Projektowanie lekcji CAI

Jak w każdej dziedzinie, również przy opracowywaniu systemu CAI należy mieć na uwadze określone zasady projektowania. Można wyróżnić trzy podstawowe aspekty projektu CAI: funkcja, cechy fizyczne i cechy logiczne.

● **Funkcja**

Dobór formy CAI i użytych środków jest uzależniony od przeznaczenia programu. Istotne jest na przykład, czy lekcja CAI prezentuje nowy materiał, czy ma charakter uzupełniający do innych metod nauczania, czyli jest rodzajem repetytorium.

Repetytorium należy do grupy programów o najmniejszych wymaganiach dotyczących reguł projektowania. Służy do ugruntowania wybranych działów wiedzy. W zasadzie mogą to być wyrywkowo potraktowane partie materiału uzupełniające podstawowy kurs dydaktyczny. Jedynym istotnym wymaganiem jest konieczność zgrania terminologii z podstawową formą dydaktyczną.

Tutor jest programem uczącym. Przekazuje on umiejętności dydaktyczne dobrego nauczyciela. Taki program powinien spełniać dwie podstawowe funkcje: prezentację nowego materiału oraz indywidualizację procesu nauczania.

Gra jest formą uatrakcyjnienia lekcji CAI. Spełnia bardzo ważną funkcję dydaktyczną, przez wykorzystanie skłonności ludzkich do hazardu. Funkcja gry może być wykorzystana w dowolnym programie, ale nie jest to zadanie proste. Opracowanie CAI z elementami gry wymaga od projektanta systemu najwyższych kwalifikacji informatycznych.

Rozwiązywanie problemów określa funkcję programów uczących dla tematów, w których istotą ma być poznanie metody rozwiązywania zadań. System CAI ma prowadzić proces dydaktyczny *krok po kroku*, aż do uzyskania rozwiązania. Co pewien czas powinno być stawiane pytanie dla oceny stopnia zrozumienia metody rozwiązywania problemu. Popełniane błędy są komentowane, po czym następuje powrót do miejsca przerwania tematu.

● **Cechy fizyczne**

Cechy fizyczne CAI określają sposób przechodzenia przez kolejne fazy jednostki lekcyjnej.

Liniowość oznacza przechodzenie przez kolejne obrazy jeden po drugim, na zasadach prostej instrukcji. Jest to cecha programów wykonywanych przez początkujących projektantów CAI lub programów, których realizacja jest ograniczona przez parametry komputera. Istotną wadą lekcji liniowej jest niemożliwość urozmaicenia programu. Każdy uczeń (student) przechodzi przez lekcję dokładnie tą samą drogą. Liniowość ma oczywiście pewne zalety: łatwość dokonywania zmian w programie CAI, nieskomplikowany algorytm, prosta obsługa programu itp.

Spiralność jest rodzajem liniowości, przy czym stopień złożoności przedstawianego materiału stopniowo wzrasta, poczynając od prostych elementów, a kończąc na złożonych układach.

Rozwidlenia oznaczają wielowariantową realizację lekcji CAI. Tej różnorodności dróg nie należy mylić z rozwidleniami programów (w sensie informatycznym), bo takie rozwidlenia występują również w liniowym modelu CAI, przy wywoływaniu stałych procedur programowych. Zasady realizacji rozwidleń CAI są ściśle związane z opisanymi metodami wartościowania odpowiedzi. Rozwidlenia są podstawą *indywidualizacji procesu nauczania*.

Wielopoziomowość jest warunkiem niezbędnym do realizacji zasady konwersacji z komputerem. Kilka poziomów informacji pozwala zindywidualizować lekcje znacznie bardziej niż rozwidlenia. W zasadzie można przyjąć założenie, że właściwe wykorzystanie cechy rozwidleń CAI jest uwarunkowane realizacją wielopoziomowości systemu. Poziom najwyższy lekcji zawiera skrótowy opis tematu z niewielką liczbą przykładów i komentarzy. Poziom najniższy zawiera oprócz treści podstawowej liczne komentarze i przykłady.

● **Cechy logiczne**

Projektując system CAI należy mieć również na uwadze pewne cechy, które wywodzą się z zasad psychologii nauczania.

Dydaktyczność polega na wzajemnej wymianie niewielkich porcji informacji między komputerem a użytkownikiem systemu. Uczeń (student) wykonuje elementarne operacje na określonej jednostce informacji i przekazuje dane do pamięci komputera.

Przykład-Formuła polega na logicznym następstwie informacji, w którym, poczynając od prostych przykładów poprzez coraz bardziej złożone, zmierzamy do zdefiniowania formuł o ogólnym znaczeniu.

Formuła-Przykład ma działanie odwrotne do poprzedniego. Kolejne dane następują w miarę poznawania elementów analizowanej formuły.

CHARAKTERYSTYKA OGÓLNA LEKCJI CAI

Pierwszym problemem, który musi rozstrzygnąć projektant CAI, jest *funkcja lekcji*.

Zgodnie z podanymi cechami lekcji CAI dla nowego materiału zaleca się formę *tutora* – uczącą. Gdy CAI ma mieć charakter uzupełnienia do innych form nauczania, projektant systemu powinien opracować lekcję jako *repetytorium*. Jeśli materiał jest łatwy do nauczania, ale trudny do zapamiętania, lub jeśli krótki czas reakcji jest istotą umiejętności, to jako podstawową formę lekcji CAI, zaleca się *repetytorium* lub *grę*.

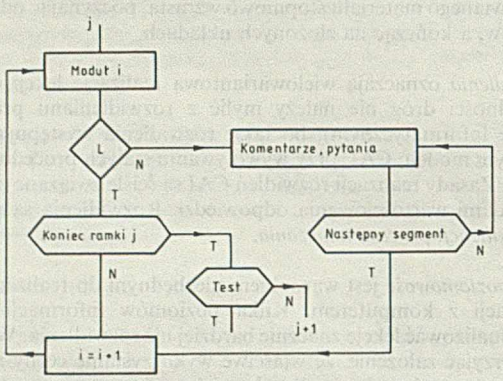
Drugim problemem rozstrzyganym przez projektanta systemu są *cechy fizyczne* CAI. Jest to jedno z najtrudniejszych zadań projektanta. Niezbędna jest bardzo dobra znajomość charakterystyk informatycznych komputera, dla którego jest opracowywany program. Zakłada się, że projektant jest dostatecznie ambitny i zamierza zrealizować najbar-

dziej złożony (najbardziej inteligentny) model CAI z wielopoziomową strukturą danych. Może się jednak okazać, że jego zamierzenia nie mogą być zrealizowane za pomocą komputera, którym dysponuje. Mogą się również pojawić inne ograniczenia, np. czas wyszukiwania danych jest zbyt długi lub niezbędna jest manipulacja dyskietkami w trakcie realizacji lekcji. Tego rodzaju niedogodności mogą rozstrzygnąć o ostatecznych cechach fizycznych zrealizowanego modelu CAI.

Problemem najtrudniejszym do usystematyzowania są reguły wyboru *cech logicznych* lekcji. Można w zasadzie stwierdzić, że takich reguł nie ma.

CAI jako system informacyjny

Opracowanie lekcji wspomaganej komputerem rozpoczyna się od *scenariusza jednostki lekcyjnej*. Informacja jest przekazywana na ekran monitora porcjami, tworząc zbiór obrazów. Każdy obraz ma swój numer identyfikacyjny (numer kolejny j), umożliwiający wyszukanie go w ciągu – jeden po drugim lub wyrywkowo. Poruszanie się w obrębie lekcji nadzoruje program *zarządzający*. Numerowany obraz składa się z kilku *faz* (warstw, *segmentów*), a jego ostateczna postać (obraz z numerem j) jest zwana *ramką*.



Rys. 1. Organizacja j-tej ramki

Każdy program CAI składa się z jednostek, a te zawierają zbiór ramek opisujących wybrane problemy. Przykładowa organizacja jednostki CAI jest następująca (rys. 1):
– każda jednostka jest podzielona na kilka (np. 10) ramek, tj. końcowych obrazów (ekranów);
– każda ramka jest podzielona na i (od 1 do k) modułów; gdy $i = 1$, mówimy o ramce 1-modułowej, a gdy $i > 1$ – o ramce wielomodułowej;
– realizacja każdej z ramek może przebiegać liniowo (szybciej) lub w sposób rozgałęziony (dłużej); literą L oznaczono blok decyzyjny wyboru wersji liniowej lub rozgałęzionej;
– w wersji rozgałęzionej każdy moduł jest zamykany prostym zadaniem; popełnione błędy są komentowane i sugerowane są partie materiału do powtórki;
– po przejściu wszystkich modułów jednostki j przechodzi się do jednostki $j + 1$;
– w wersji liniowej przewiduje się przejście do realizacji testu po wykonaniu programu zawartego we wszystkich ramkach jednostki, choć jest również możliwe przerwanie programu i uruchomienie testu po każdej ramce (z klawiatury komputera nieautomatycznie).

Zastosowanie mikrokomputera 8-bitowego o pamięci operacyjnej nie przekraczającej 64 KB stwarza niewielkie możliwości rozwidleń programu. Nawet stacja dysków elastycznych (360 KB) nie wystarcza do przechowania informacji merytorycznej oraz programu zarządzającego CAI.

Wyróżnia się dwie podstawowe kategorie ramek CAI: *uczącą* i *diagnostyczną*. Zdefiniowanie głównych cech tych ramek jest bardzo przydatne w trakcie projektowania, realizacji, sprawdzania i oceny wartości lekcji CAI.

Ramka ucząca zawiera podstawową informację dla użytkownika CAI, w formie:
– *informacyjnej*, z prezentacją informacji statycznej (napisy, plansze),
– *symulacyjnej*, objaśniającej zasady działania układu lub urządzenia.

Prostsza w realizacji jest oczywiście forma informacyjna. Forma symulacyjna jest rozszerzona o elementy animacji obrazu, co zwiększa wartość tego rozwiązania. Dopuszczalne jest wzajemne przeplatanie rodzajów ramek uczących.

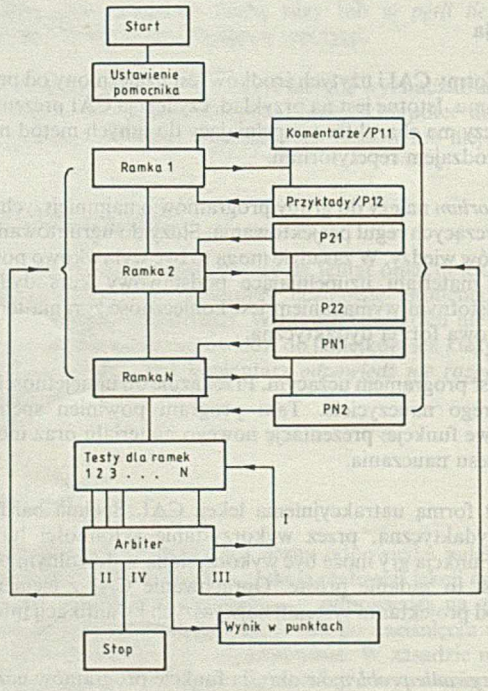
Ramka diagnostyczna służy do sprawdzania stopnia opanowania wiadomości zaprezentowanych w ramkach uczących. Wartościowanie odpowiedzi ćwiczącego odbywa się według kilku utartych wzorów; są nimi ramki:
– *alternatywnych odpowiedzi:* jako rozwiązanie najprostsze (o cechach loterii) nie jest zalecane w systemach CAI;
– *uzupełniania odpowiedzi*, gdzie komputer przedstawia część odpowiedzi (w celu wykluczenia niejednoznaczności), która zostaje uzupełniona przez ćwiczącego;
– *wyboru jednej z dróg*, zalecane w przypadkach odpowiadania na pytania o długim ciągu następstw, gdzie odpowiedź ma wiele kolejno następujących po sobie ogniw;
– *sekwencji odpowiedzi*, dla zagadnień, w których istnieje jednoznaczny ciąg odpowiedzi cząstkowych stanowiący odpowiedź finalną;
– *tablicy odpowiedzi*, polegającej na wypełnianiu pól na zadanym obrazie (odmianie metody poprzedniej).

Stopień złożoności programu CAI zwiększa się bardzo szybko wraz ze stopniem inteligencji programu zarządzającego. Największą złożoność mają programy ekspertowe, zawierające elementy symulacji z rozwidleniami programów i mechanizmami wnioskowania.

MODEL INTELIGENTNEGO CAI

W omówionych strategiach dydaktycznych decyzja wyboru kolejnego kroku wynika z realizacji zadań w kroku poprzednim. W inteligentnej wersji programu wspomagającego proces dydaktyczny – ICAI (ang. *Intelligent* CAI) [7], równie złożony jak sam program użytkowy jest program organizacyjny, ułatwiający pracę użytkownika systemu. Faza pytań może się rozpocząć po przekazaniu uczącemu się dostatecznego zbioru wiadomości. To wymaganie określa minimalne rozmiary części uczącej i wynikające z takiego założenia minimalne rozmiary pamięci operacyjnej mikrokomputera.

W systemie opracowanym w Zakładzie Elektroniki i Informatyki Uniwersytetu Śląskiego wybrane działy problemowe podzielono na trzy jednostki uczące i jedną jednostkę sprawdzającą poziom wiedzy. W wyniku oceny odpowiedzi można wybrać dalsze fazy nauczania. Komputer przedstawia listę popełnionych błędów oraz sugestie dotyczące następnego kroku w realizacji programu. Zamiast oceny testu proponuje się procent poprawnych odpowiedzi.



Rys. 2. Sieć działań dla jednostki ICAI

Prowadzący zajęcia lub ćwiczący może przerwać realizację lekcji CAI w dowolnej chwili i przejść do wybranego ekranu zgodnie z wprowadzoną numeracją dziesiętną ekranów.

Program CAI powinien mieć dwie podstawowe cechy programowanego nauczania, tj. *konwersacyjność* i *wielopoziomowość*. Realizacja takiego systemu jest możliwa wówczas, gdy jest dostępny komputer o znacznej pamięci zewnętrznej i dostatecznej pamięci operacyjnej (kilkuset kilobajtów).

Na rysunku 2 przedstawiono sieć działań programu spełniającego cechy elementarnego ICAI. Każda jednostka zawiera *N* ramek (końcowych ekranów). Na zakończenie jednostki jest przeprowadzany test sprawdzający stopień opanowania tematu jednostki. Popełnione w trakcie realizacji testu błędy są rejestrowane w programie arbitra.

Na zakończenie testu jest przedstawiana lista tych błędów i numery odpowiednich ekranów, określające materiał powtórki.

Zadaniem arbitra jest analiza:
- liczby popełnionych błędów,
- krotności wykonania błędu testu.

Wyróżniono cztery poziomy błędów:
I – nieznaczna liczba błędów (ponad 80% poprawnych odpowiedzi); proponuje się powtórkę testu (z podaniem listy błędów);
II – duża liczba błędów popełnionych po raz pierwszy (ocena dostateczna); proponuje się powtórzenie treści ramek;
III – duża liczba błędów popełnionych po raz drugi (ocena niedostateczna); oferuje się komentarze uzupełniające i przykłady do treści odpowiadających błędnym odpowiedziom;
IV – błąd popełniony trzeci raz; określa negatywny wynik nauczania.

Analizując cechy zaproponowanego modelu CAI, można w nim znaleźć wiele cech strategii Crowdera. W razie popełnienia błędów następuje powrót do fazy instruowania z możliwością wywołania dodatkowych komentarzy. Można tu również zauważyć elementy strategii Kaya, ponieważ przewidziano wstępne ustawianie poziomu pomocy, a więc dobieranie poziomu szczegółowości lekcji.

ETAPY PROJEKTOWANIA SYSTEMÓW CAI

Etap I
1. Wyodrębnienie zagadnień węzłowych przedmiotu o dostatecznie atrakcyjnej treści graficznej (proponowanego obrazu).
2. Wybranie form realizacji lekcji (opis, demonstracja czy symulacja komputerowa).
3. Wytypowanie zagadnień węzłowych, z punktu widzenia cech systemu CAI – do realizacji symulacji komputerowej (ocena charakterystyk komputera, ocena „podatności komputerowej" zagadnienia).

Etap II
4. Opracowanie wstępnego scenariusza jednostki programowej.
5. Analiza form sprawdzania opanowania problematyki jednostki.
6. Korekta scenariusza na podstawie prac w punkcie 5.

Etap III
7. Opracowanie obrazów statycznych, tzw. ramek, z rozplanowaniem szaty graficznej obrazu, z uwzględnieniem zasad minimalizacji tekstów komentarza i czytelności opisów, cech estetycznych prezentowanego obrazu itp.
8. Analiza niezbędnej dynamiki do ilustracji obrazu w obrębie ramki oraz zakresu komentarzy w poszczególnych fazach pośrednich ramki.
9. Wybranie form sprawdzenia opanowania treści jednostek – dobór treści zdań i pytań, określenie sposobów oszacowania odpowiedzi itp.

Etap IV
10. Opracowanie założeń do programu zarządzającego blokiem programowym, tj. sposobu realizacji przerwań, skoków „do przodu" i „do tyłu" w programie, zasad wnioskowania komputera, zasad doboru poziomów pomocy.

Etap V
11. Opracowanie prototypu programu CAI.
12. Wstępna weryfikacja programu w procesie dydaktycznym; zebranie uwag merytorycznych i metodycznych na temat opracowanego programu.
13. Korekty merytoryczne i metodyczne programów CAI.
14. Opracowanie podręcznika dla ucznia.

Etap VI
15. Weryfikacja wartości merytorycznych i metodycznych opracowanego prototypu programu w rzeczywistych warunkach dydaktycznych.
16. Opracowanie materiałów pomocniczych do programów (przewodnik metodyczny dla nauczyciela) oraz materiałów źródłowych dla dystrybutora systemu CAI.

Zasada oceny wartości lekcji

Opracowany system nauczania przed przekazaniem do eksploatacji należy poddać *ocenie wartości*, tzn. jego przydatności do realizacji założonego celu dydaktycznego. Prototyp programu CAI przekazuje się współpracownikom lub innym projektantom do przeprowadzenia wstępnej oceny merytorycznej i metodycznej. Wyróżnić tu można trzy etapy.

Etap I, w którym przedmiotem oceny są:
- zgodność treści lekcji z programem nauczania przedmiotu,
- wartość merytoryczna lekcji,
- walory graficzne programu,
- sposób i czas ekspozycji treści programowych (tempo realizacji lekcji),
- forma pytań i komentarze odpowiedzi,
- komunikatywność dyrektyw operatorskich,
- walory użytkowe programu organizacyjnego (poruszanie się w obrębie jednostki i bloku).

Recenzent wypełnia arkusz oceny merytoryczno-metodycznej, według wzoru podanego w tabeli 1. Do arkusza powinna zostać dołączona lista uwag szczegółowych.

Tabela 1. Arkusz oceny merytoryczno-metodycznej

Pozycja	Nr Bloku	Nr Jednostki	Nr Ramki	Uwagi
Brak zgodności z programem				
Błędy merytoryczne				
Forma graficzna – błędna – niezrozumiała				
...				
Nieefektywny program organizacyjny				
Uwagi szczegółowe:				

Etap II, w którym celem oceny są:
- sprawdzenie komunikatywności eksponowanych treści w jednostkach uczących,
- sprawdzenie jednoznaczności pytań jednostki sprawdzającej,
- sprawdzenie średniego czasu trwania jednostki w pracowni szkolnej,
- opracowanie podręcznika ucznia.

Nauczyciel wypełnia arkusz przebiegu eksperymentu według wzoru przedstawionego w tabeli 2.

Ocena sprawdzianu powinna odpowiadać *kryterium* 90/90, tzn. ocena materiału CAI jest pozytywna, gdy 90% reprezentatywnej grupy uczniów (studentów) zrozumie za pierwszym razem (bez dodatkowych objaśnień) 90% materiału lekcji CAI. Badania te muszą być przeprowadzone w warunkach identycznych do warunków przyszłej eksploatacji systemu CAI.

Jeśli wyniki testu są zbyt pozytywne, to można przyjąć założenie, że problematyka lekcji lub pytania są zbyt łatwe. W przypadku wielu gorszych odpowiedzi należy przyjąć, że lekcja lub pytania wymagają poprawek. Jeśli kolejne poprawki nie przynoszą pozytywnych rezultatów, to należy gruntownie zmienić koncepcję programu.

Tabela 2. Arkusz przebiegu eksperymentu

Pozycja	Nr Bloku	Nr Jednostki	Nr Ramki	Uwagi
Niezrozumiała treść				
Pytania niejednoznaczne				
Czas trwania				
Nie spełnione kryterium 90/90				

Uwagi szczegółowe:

Bardzo przydatnym uzupełnieniem oceny jest pomiar czasu odpowiedzi; dobrej i złej. Statystycznie szybka, ale zła odpowiedź sugeruje nieuwagę ćwiczącego. Może być jednak świadectwem źle sformułowanego pytania, które sugeruje niewłaściwą odpowiedź. Gdy odpowiedź jest dobra, ale jej udzielenie zabiera zbyt dużo czasu, to oznacza to, że segment lekcji związany z tym pytaniem wymaga uzupełnień.

Trzecim etapem oceny CAI są tzw. *badania polowe* (ang. *Field testing*) na lekcjach eksperymentalnych.

Etap III, w którym celem oceny są:
– opracowanie metody nauczania dla proponowanego systemu CAI (wymiar godzin, praca z nauczycielem – praca własna), jako forma wdrożenia systemu,
– opracowanie przewodnika metodycznego dla nauczyciela,
– ocena efektywności nauczania.

BADANIA EMPIRYCZNE EFEKTYWNOŚCI NAUCZANIA

Ostatnim etapem badań jest ocena efektów nauczania. Ten etap badać ma zdecydowanie naukowy charakter i nie może być wymagany dla każdej lekcji. Wynika to z dużego kosztu oraz długiego czasu trwania badań empirycznych.

Do oceny wartości komputerowych programów wspomagających nauczanie proponuje się wykorzystać: *metodę eksperymentu pedagogicznego* lub *metodę testów pedagogicznych*. Jako uzupełnienie można zastosować *metodę obserwacji* [4].

Eksperyment pedagogiczny jest jedną z podstawowych form badań pedagogicznych. W założeniach prac teoretycznych znajduje się wiele sformułowań określających sposób i warunki badań. Niestety, większość z nich ma charakter nic nie mówiących ogólników.

Wydaje się, że powodzenie eksperymentów zależy od tego, czy:
– CAI obejmuje dostatecznie duży materiał dydaktyczny,
– istnieją sposoby pomiaru uzyskanych efektów.

Pierwszy warunek kwestionuje wartość rezultatów badań wykonanych na niewielkim materiale eksperymentalnym. Odpowiedź jest oczywista. Badania, w których dokonuje się oszacowań statystycznych, są tym więcej warte, im obszerniejszy jest materiał badawczy. Dlatego nie można traktować serio eksperymentu pedagogicznego przeprowadzanego na jednej lekcji.

Warunek drugi jest związany z wyborem techniki prowadzenia eksperymentu. Najprostsza organizacyjnie jest technika grup równoległych (porównawczych). Wybiera się dwie grupy badawcze: eksperymentalną, nauczaną przy użyciu badanego systemu CAI, oraz grupę porównawczą, nauczaną tradycyjnie (tę samą problematykę).

Autorzy metod badań pedagogicznych warunkują ich wartość od spełnienia wielu „nieprawdopodobnych" warunków, dlatego – zdaniem autora – wyniki takich ocen nie mają wartości wymiernej, a jedynie orientacyjną.

Znacznie prostsze w realizacji są badania wartości CAI metodami obserwacyjnymi. Również w tej dziedzinie proponuje się wiele metod, z których godną polecenia jest najprostsza – *technika obserwacji uczestniczącej*. Metodę obserwacyjną można stosować jako uzupełnienie metody badań pedagogicznych. Może to również być jedyna metoda oceny wartości dydaktycznych materiałów CAI o niewielkim wymiarze merytorycznym.

Na zakończenie badań warto przeprowadzić *sondaż diagnostyczny* w celu zebrania opinii uczniów (studentów) i nauczyciela na temat systemu CAI wykorzystanego w lekcjach. Chodzi przede wszystkim o zlokalizowanie elementów najtrudniejszych, wskazanie niejednoznacznej treści i pytań oraz o ogólne wrażenia metodyczne na temat materiału prezentowanego w systemie.

* * *

Przedstawiona powyżej dyskusja na temat cech, ocen wartości oraz zasad projektowania systemów CAI ma charakter wniosków z wieloletnich prac zespołu badawczego pracowników Uniwersytetu Śląskiego. Inne aspekty tych badań zostaną przedstawione w kolejnych opracowaniach, po zakończeniu dalszych eksperymentów dydaktycznych. Na przykład, godnym uwagi jest język obiektowy opracowany w Zakładzie Elektroniki i Informatyki, służący do projektowania systemów CAI, przez osoby bez znajomości typowych języków programowania komputerów, takich jak Pascal czy Basic.

LITERATURA

[1] Burke R.L.: CAI Source Book. Prentice-Hall, New Jersey, 1982
[2] Erberts R.E.: Learning strategies in CAI design. The International Journal of Applied Engineering Education, No 2, Oxford, 1986
[3] Godlewski M., Kołkowski L., Lipiński E.: Problemy algorytmizacji i automatyzacji procesu dydaktycznego. PWN, Warszawa, 1983
[4] Łobocki M.: Metody badań pedagogicznych. PWN, Warszawa, 1984
[5] Computers in Education. Proc. of the IFIP 2-nd Word Conference. North-Holland Publishing Company, Amsterdam, 1975
[6] Piecha J.: The CAI microcomputer system. Some problems with strategy and memory reduction. Proc. of 10th Int. Conf. Computer at the University, pp. 1.15.1–1.15.8, Dubrovnik – Yugoslavia, June 1988 (Wissenschaftliche Zeitschriften der Pedagogische Hochschule Erfurt-Mulhausen. Matematisch-Naturwissenschaftliche Reihe, No. 3, 1988)
[7] Piecha J.: The multilevel model for microcomputer ICAI systems. The international Journal of Applied Engineering Education. Vol. 5, No. 3. Pergamon Press, Oxford 1989 (Proc. of 11th Int. Conf. Computer at the University, Dubrovnik – Yugoslavia, June 1989)
[8] Piecha J.: Strategie dydaktyczne a parametry mikrokomputera w systemie CAI. Zeszyt Naukowy „Mikrokomputery w Procesie Dydaktycznym". Centrum Doskonalenia Nauczycieli, Katowice, 1989
[9] Piecha J.: General restrictions for computers in education. Wissenschaftliche Zeitschriften der Padagogische Hochschule Erfurt-Mulhausen, Matematisch-Naturwissenschaftlische Reihe, Erfurt, 1989.

Okazało się, że mimo ubiegłorocznej podwyżki ceny nie udało się nam wymknąć spod żelaznych reguł rynku, zmuszających do podwojenia w tym roku ceny czasopisma. Jedynym pocieszeniem jest fakt, że we wzroście ceny nie byliśmy odosobnieni, oraz, że w wyścigu tym nie zajmujemy czołowego miejsca. Czujemy się w obowiązku poinformowania Czytelników o podstawowych przyczynach tej tak przykrej dla nas decyzji.

Przyczyną podstawową był oczywiście szybki wzrost wszystkich składników kosztów wydawniczych jako konsekwencji postępującej inflacji. Nie bez znaczenia było jednak zmniejszenie się liczby prenumeratorów. Zmniejszenie to wynikło z głębokiego kryzysu, jaki przeżywają instytucje budżetowe, a zwłaszcza placówki biblioteczne, które stanowiły dotąd znaczny odsetek prenumeratorów. Ubytek ten powinny wprawdzie z nadwyżką zrekompensować tak licznie powstające firmy informatyczne, jednak wskutek braku odpowiedniej informacji wydawniczej, do tego szerokiego kręgu potencjalnych odbiorców czasopisma nie dociera jeszcze fakt, że już dwudziesty szósty rok INFORMATYKA pełni funkcję prekursora w informowaniu o postępie w technologii informatycznej.

Redakcja podejmuje działania zmierzające do zwiększenia liczby prenumeratorów, a więc zwiększenia nakładu, co najskuteczniej powinno przeciwdziałać wzrostowi ceny. Liczymy, że w działaniach tych pomogą nam również nasi najwierniejsi Czytelnicy.

Redakcja

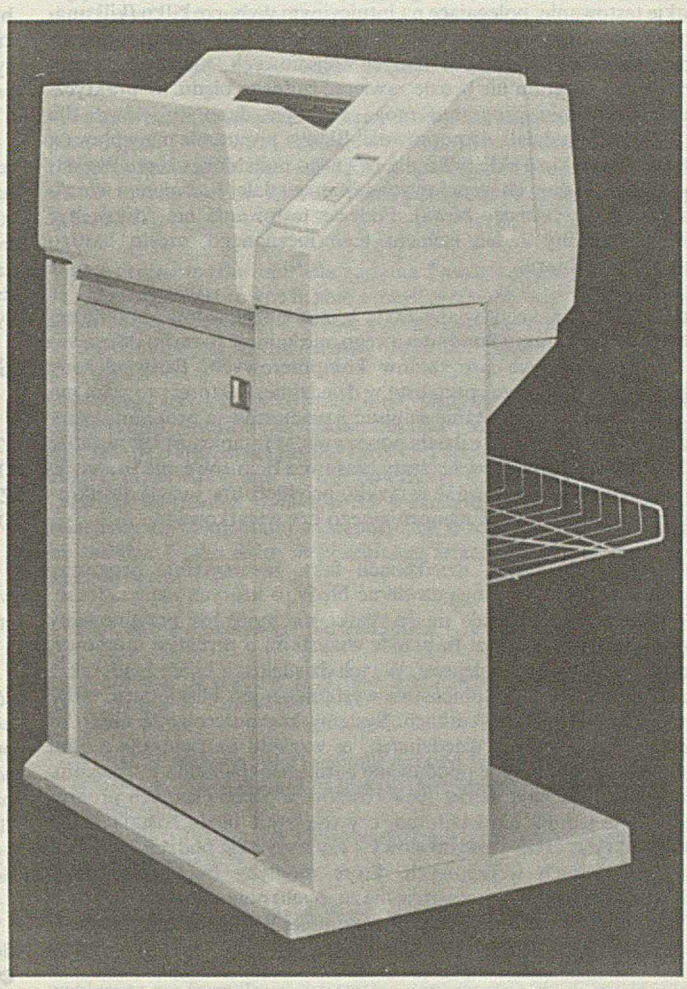
SUPRA SP. Z O.O.
ul. Trybunalska 54, 60-325 POZNAŃ
telefon: 674392, telefaks: 236498
teleks: 412966

OFERUJE ŚWIATOWY BESTSELLER

MANNESMANN TALLY

DRUKARKĘ MOZAIKOWO-WIERSZOWĄ MT 645

- doskonała jakość druku
- niezawodność 7000 h MTBF
- duża szybkość 450 wierszy/min
- cicha praca
- bufor 27 KB
- liczba kopii 1+5
- pełna grafika 240 (dpi)
- polskie litery (opcja)
- wewnętrzny program testujący
- dwa interfejsy automatycznie przełączane
- podwójny system traktorów



MANNESMANN TALLY

NAJPRAWDOPODOBNIEJ NAJLEPSZE

DRUKARKI NA ŚWIECIE

AUTORYZOWANY DYSTRYBUTOR DRUKAREK

MANNESMAN TALLY

0/12/90

DOROTA KUCHTA
Instytut Organizacji i Zarządzania
Politechnika Wrocławska

Systematyczne testowanie programów zalety i wady (1)

Każdy piszący program stosuje testowanie: wybiera różne dane wejściowe i sprawdza, czy napisany program „chodzi" dla tych danych i czy daje oczekiwane wyniki. Takie sprawdzanie programu kontynuuje się tak długo, aż programista nabierze „wewnętrznego przekonania", że jego program jest w jakimś rozsądnym stopniu poprawny i że podczas praktycznej eksploatacji będzie dawał z dużym prawdopodobieństwem poprawne wyniki.

Takie testowanie, polegające na intuicyjnym wyborze kilku (kilkunastu) sprawdzających danych wejściowych spośród zazwyczaj ogromnej liczby wszystkich możliwych danych wejściowych, nie może jednak zapewnić, że program nie będzie zawierać żadnego błędu. W praktycznym zastosowaniu programu mogą wystąpić dane wejściowe, dla których program działa niepoprawnie. Błąd w programie ma wpływ na otrzymane wyniki zwykle tylko dla pewnego podzbioru zbioru wszystkich możliwych danych wejściowych (zwanego dalej *podzbiorem wrażliwym dla rozpatrywanego błędu*). Podczas testowania nie zostaje być może sprawdzony żaden element tego nieznanego, często bardzo „małego" podzbioru.

Powszechnie zaakceptowany fakt, że nie można udowodnić pełnej poprawności programu komputerowego, nie hamuje rozwoju i stosowania coraz to nowych programów komputerowych. Doświadczony programista wspólnie ze specjalistą w dziedzinie, w której program ma być zastosowany, są w stanie napisać i przetestować program, który rzeczywiście prawie zawsze działa poprawnie. Wystąpienie błędów może być kłopotliwe i spowodować straty czasowe i finansowe, ale trudności tego rodzaju są wkalkulowane w ryzyko producentów i użytkowników i nie uważa się ich za coś nienormalnego czy wyjątkowego.

W niektórych jednak dziedzinach fakt, że wszystkie programy stosowane w praktyce mogą zawierać błędy, o których nie wiadomo, kiedy wystąpią i jakie będą miały skutki, nie może być przyjmowany z takim spokojem. Chodzi tu przede wszystkim o przemysł atomowy, komunikację lotniczą i kolejową. W tych dziedzinach każdy błąd, także taki, którego prawdopodobieństwo wystąpienia jest bliskie zeru, może być tragiczny w swoich skutkach. Systemy komputerowe są mimo to stosowane także w tych dziedzinach, ze względu na związane z nimi ogromne korzyści. Są one poddawane bardzo dokładnemu testowaniu, którego koszty sięgają nieraz 70% kosztów napisania samych programów. Ciągle jednak nie wiadomo, czy wszystkie błędy zostają w ten sposób wykryte, czyli – innymi słowy – czy wszystkie podzbiory zbioru wszystkich danych wejściowych, które mogłyby być podzbiorami wrażliwymi dla błędów (oczywiście nie znanych) popełnionych w danym systemie, miały przy testowaniu przynajmniej jednego swojego przedstawiciela.

Osiągnięcie tego celu próbuje się przybliżyć przez systematyzację testowania. Oznacza ona, że przypadki testowe nie są wybierane intuicyjnie, lecz według formalnych reguł, zapewniających pokrycie (w sensie, który będzie w dalszym ciągu wyjaśniony) przypadkami testowymi całości napisanego programu lub (i) zadania. Pokrycia te oznaczają formalne podziały zbioru wszystkich danych wejściowych na podzbiory. Ma się nadzieję, że te podzbiory, związane ściśle z napisanym programem i z rozwiązywanym zadaniem, są również w jakiś sposób związane z błędami, które można popełnić przy programowaniu rozpatrywanego systemu i dzięki temu doprowadzą do systematycznego wychwycenia podzbiorów wrażliwych.

Nadzieje takie opierają się na optymistycznych ocenach systematycznych metod testowania często spotykanych w literaturze (np. [1]). Niestety, oceny te są zwykle nieformalne, wynikają często z rozumowania typu *„metoda jest systematyczna, a więc dobra"*. Systematyczne metody testowania rzeczywiście mają wiele zalet, co jest w literaturze dotyczącej weryfikacji programów poparte wieloma przykładami, ale nie zapewniają wykrycia nawet najczęściej spotykanych, najłatwiejszych do popełnienia typów błędów. Systematyzacja testowania prowadzi wprawdzie prawie zawsze do zwiększenia prawdopodobieństwa wykrycia błędów w stosunku do testowania intuicyjnego (przypadkowego), ale rzadko do wartości bliskiej 1, przy czym dla niektórych typów błędów prawdopodobieństwo to po systematyzaji nie ulega istotnej zmianie lub nawet zmniejsza się. Między poszczególnymi metodami systematycznymi występują znaczne różnice w ich możliwościach wykrywania błędów.

W pierwszej części artykułu opisano najbardziej znane i najczęściej używane systematyczne metody testowania, a ich użycie zilustrowano na kilku przykładach programów. W drugiej części zostaną przedyskutowane ich wady i zalety, ze szczególnym podkreśleniem wad – niemożności zapewnienia wykrycia wielu typów błędów. Zostaną również zasygnalizowane praktyczne możliwości poprawy zdolności testowania do wykrywania błędów przez modyfikacje znanych metod testowania systematycznego.

Systematyczne metody testowania – definicja

Niech będą dane dwa zadania:
Zadanie 1
Obliczyć maksimum dwóch liczb rzeczywistych *a* i *b*.
Zadanie 2
Obliczyć maksimum trzech liczb rzeczywistych *a, b, c*.

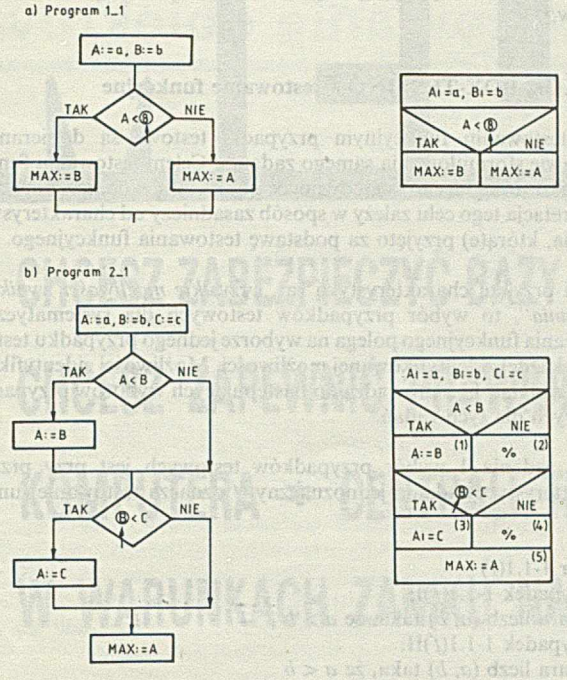
Niżej przedstawiono dwa programy, po jednym dla każdego z powyższych zadań (rys. 1). Oba programy są błędne: błąd zaznaczono strzałką i kółkiem. Poprawne programy przedstawiono na rysunku 2.

Oba programy są przedstawione w dwóch, w tym przypadku równoważnych postaciach: za pomocą schematów blokowych oraz za pomocą mniej znanych diagramów strukturalnych. Te ostatnie znajdują powszechne zastosowanie w konstrukcji systemów komputerowych w niektórych dziedzinach, np. w kolejnictwie. Uważa się, że zmuszają one do większej dyscypliny podczas programowania (programowanie strukturalne), dzięki czemu wiele błędów zostaje od początku wyeliminowanych, oraz zapewniają większą czytelność programów i ich dokumentacji.

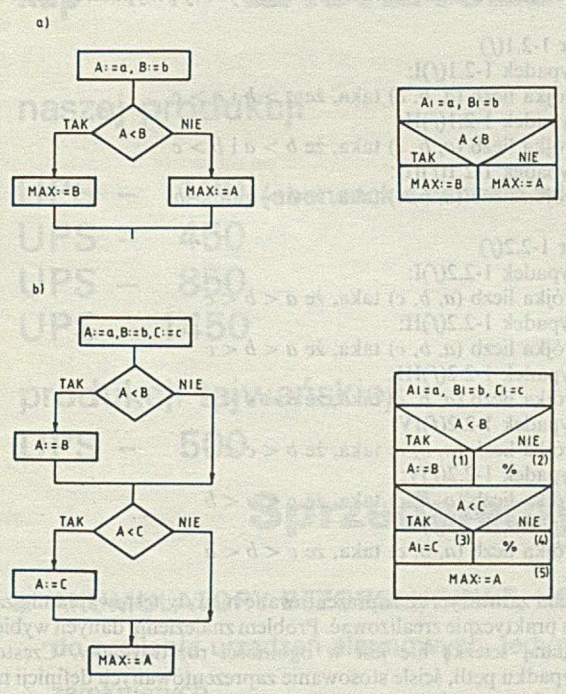
Do zdefiniowania metod testowania są potrzebne m.in. dwa pojęcia dotyczące struktury programu:
- *gałązka:* pojedyncza instrukcja lub sekwencja instrukcji nie będących instrukcjami warunkowymi, np. w programie 2_1 każdy z prostokątów (1), (2), ..., (5);
- *ścieżka:* dowolny zbiór gałązek, stanowiący drogę od początku do końca programu; w programie 2_2 istnieją 4 ścieżki: (1)(3)(5), (1)(4)(5), (2)(3)(5), (2)(4)(5), a w programie 1_1 tylko dwie.

Struktura programu jest charakterystyką programu, którą można formalnie definiować, wyróżniając pewne jej elementy (np. oba wyżej wspomniane). Na tej podstawie można definiować pewne systematyczne metody testowania. Wspomniane we wstępie *„pokrycie przypadkami testowymi całości programu"* będzie tu oznaczać pokrycie poszczególnych elementów struktury programu. Okazuje się jednak, że testowanie systematyczne oparte tylko na strukturze programu nie wystarczy do

zapewnienia dobrej wykrywalności błędów w programie. Potrzebne jest również testowanie systematyczne oparte na samym zadaniu, jego sformułowaniu, realizujące *„pokrycie przypadkami testowymi całości zadania"*.



Rys. 1. Przykłady błędów popełnionych w programach dla zadań 1 i 2



Rys. 2. Przykłady poprawnych programów dla zadań 1 i 2

Tutaj trudniej jest znaleźć formalnie definiowalne charakterystyki mogące stanowić podstawę definicji *„pokrycia zadania"*. Ważne jest przy tym, żeby charakterystyki te dotyczyły całości zadania, czyli rzeczywiście w pewien sposób *„pokrywały"* całe zadanie, tak jak struktura programu odnosi się do całości programu. Trudności w znalezieniu formalnych charakterystyk pokrywających zadanie wynikają przede wszystkim stąd, że zadania zwykle nie są dane w żadnej jednolitej, uniwersalnej i formalnej postaci. Wciąż jeszcze przeważają sformułowania testowe (choć w niektórych dziedzinach spotyka się już coraz częściej zadania w języku półformalnym). Istnieje wiele podejść do znalezienia pewnych charakterystyk zadań dla programów komputerowych, które mogłyby służyć za podstawę definicji systematycznych metod testowania. Nie są one niestety tak uniwersalne, jak np. pojęcie struktury programu, ponieważ zależą w dużym stopniu od dziedziny, której dotyczy zadanie – inne charakterystyki są rozpatrywane przy problemach obliczeniowych (przykładami takich charakterystyk są wartości ekstremalne lub liczba miejsc zerowych rozpatrywanych funkcji, końce przedziałów wartości wyróżnionych w zadaniu), a inne przy problemach nie dotyczących bezpośrednio skomplikowanych obliczeń na liczbach rzeczywistych, lecz np. polegających na podjęciu pewnej decyzji na podstawie istniejących warunków lub dostępnych danych.

W problemach tego rodzaju rozpatruje się m.in. charakterystykę *„wszystkie możliwości, układy wynikające z zadania"*. Jest to charakterystyka nieformalna, w wielu przypadkach nie jest jasne, co to pojęcie oznacza. Mimo to, metody testowania systematycznego oparte na tej charakterystyce zadań są często stosowane w praktyce. Właśnie te metody testowania systematycznego będą – obok metod testowania systematycznego opartych na strukturze programu – przedmiotem rozważań w tym artykule. Zostaną one zilustrowane na przykładzie zadań 1 i 2.

Intuicyjna interpretacja charakterystyki *„wszystkie możliwości wynikające z zadania"* dla zadania 1 wydaje się jednoznaczna. Oto *„różne możliwości"* dla zadania 1:
1) maksimum jest równe liczbie *a* (czyli pierwszej wczytywanej liczbie);
2) maksimum jest równe liczbie *b* (czyli drugiej wczytywanej liczbie).

Jednak już przy niewiele bardziej skomplikowanym zadaniu 2 wynik poszukiwań *„różnych możliwości wynikających z zadania"* nie musi być tak jednoznaczny. Najbardziej prawdopodobne wydaje się zidentyfikowanie możliwości w sposób analogiczny do rozpatrywanego przy zadaniu 1.

Oto *„różne możliwości"* dla zadania 2 (wariant pierwszy):
1) maksimum jest równe liczbie *a* (czyli pierwszej wczytywanej liczbie);
2) maksimum jest równe liczbie *b* (czyli drugiej wczytywanej liczbie);
3) maksimum jest równe liczbie *c* (czyli trzeciej wczytywanej liczbie).

Możliwa jest jednak również inna, bardziej formalna interpretacja *„możliwości"*, która w przypadku zadania 1 nie wprowadziłaby wprawdzie żadnych zmian, ale już dla zadania 2 spowodowałaby wyznaczenie zbioru możliwości innego niż powyższy. Chodzi tu o interpretację *„różnych możliwości wynikających z zadania"* jako różnych układów relacji między elementami występującymi w zadaniu, a więc w przypadku zadania 2 jako różnych kombinacji relacji $<$, $>$ między liczbami rzeczywistymi *a*, *b*, *c*.

Oto *„różne możliwości"* dla zadania 2 (wariant drugi):

1) $a < b < c$, 2) $a < c < b$, 3) $b < a < c$, 4) $b < c < a$, 5) $c < a < b$, 6) $c < b < a$

Wyróżnia się dwa typy systematycznego testowania:
– *White-Box-Testing*, czyli testowanie strukturalne (oparte na strukturze programu);
– *Black-Box-Testing*, czyli testowanie funkcyjne (oparte na charakterystykach zadania).

Podczas testowania pierwszego typu sprawdzany program jest traktowany jako *„white box"*, czyli jako obiekt o znanej strukturze wewnętrznej (a więc np. gałązkach i ścieżkach). Przypadki testowe są wybierane wg formalnych kryteriów odnoszących się do tej struktury. Przy testowaniu drugiego typu wewnętrzna struktura sprawdzanego programu jest traktowana jako nieznana (program jest traktowany jako *„czarna skrzynka"*), przypadki testowe są dobierane na podstawie wybranych charakterystyk samego zadania i jego sformułowania.

Dotychczasowe rozważania pozwalają już teraz stwierdzić, że metody z grupy *„White-Box-Testing"* są metodami bardziej formalnie określonymi niż metody *„Black-Box-Testing"*.

„WHITE-BOX-TESTING – testowanie strukturalne

Celem testowania strukturalnego jest *„całkowite pokrycie"* struktury sprawdzanego programu przez wybrane przypadki testowe. Różne

metody testowania strukturalnego wywodzą się z różnych definicji *„całkowitego pokrycia"* struktury programu. I tak, *testowanie gałązkowe* (*branch testing*) wymaga wybrania takiej minimalnej liczby przypadków testowych, aby każda gałązka sprawdzanego programu została podczas testowania przynajmniej raz uruchomiona. Dla programu 1-1 oznacza to konieczność wyboru dwóch przypadków testowych spełniających następujące warunki (*s* oznacza testowanie strukturalne):

Wybór 1-1.1(*s*)
  Przypadek 1-1.1(*s*)I:
    para liczb (*a, b*) taka, że $a < 8$ (dla gałązki TAK)
  Przypadek 1-1.1(*s*)II:
    para liczb (*a, b*) taka, że $a >= 8$ (dla gałązki NIE)

Program 1-2 zawiera znacznie więcej (5) gałązek niż program 1-1. Fakt ten nie musi jednak oznaczać, że realizacja testowania gałązkowego dla programu 1-2 wymaga więcej przypadków testowych niż dla programu 1-1. Liczba przypadków testowych konieczna do uruchomienia przynajmniej raz każdej gałązki zależy nie tyle od liczby gałązek, ile od innych własności struktury programu. W programie 1-2 dowolny przypadek testowy wymaga uruchomienia 3 gałązek, dlatego do realizacji testowania gałązkowego dla tego programu wystarczą również dwa przypadki testowe, pod warunkiem, że będą one odpowiednio wybrane (ze względu na koszty należy starać się zrealizować kryterium pokrycia struktury jak najmniejszą liczbą przypadków testowych). Można to zrealizować na dwa sposoby.

Wybór 1-2.1(*s*)
  Przypadek 1-2.1(*s*)I:
    trójka liczb (*a, b, c*) taka, że $a < b$ i $b < c$ (gałązki (1), (3), (5))
  Przypadek 1-2.1(*s*)II:
    trójka liczb (*a, b, c*) taka, że $a >= b$ i $b >= c$ (gałązki (2), (4), (5))

Wybór 1-2.2(*s*)
  Przypadek 1-2.2(*s*)I:
    trójka liczb (*a, b, c*) taka, że $a < b$ i $b >= c$ (gałązki (1), (4), (5))
  Przypadek 1-2.2(*s*)II:
    trójka liczb (*a, b, c*) taka, że $a >= b$ i $b < c$ (gałązki (2), (3), (5))

Inną metodą z grupy *„White-Box-Testing"* jest *testowanie ścieżkowe* (ang. *path testing*): wymaga ono uruchomienia dla każdego przypadku testowego nie tylko wszystkich gałązek, ale także wszystkich ścieżek. Dla programu 1-1 testowanie ścieżkowe jest równoważne z testowaniem gałązkowym, ale dla programu 1-2 oba sposoby testowania różnią się zasadniczo: testowanie ścieżkowe wymaga tu przynajmniej czterech przypadków testowych.

Wybór 1-2.3(*s*)
  Przypadek 1-2.3(*s*) I – identyczny z 1-2.1(*s*)I
  Przypadek 1-2.3(*s*) II – identyczny z 1-2.1(*s*)II
  Przypadek 1-2.3(*s*)III – identyczny z 1-2.2(*s*)I
  Przypadek 1-2.3(*s*)IV – identyczny z 1-2.2(*s*)II

Testowania ścieżkowe i gałązkowe są najczęściej stosowanymi w praktyce metodami z grupy *„White-Box-Testing"*.

Wybór przypadków testowych 1-1.1(*s*) mógłby nasuwać wątpliwości co do celowości kontynuowania testowania (tzn. wyboru konkretnych przypadków testowych i użycia ich jako danych wejściowych dla sprawdzanego programu) po otrzymaniu warunków dla przypadków testowych 1-1.1(*s*)I i 1-1.1(*s*)II. Warunki te wydają się bowiem same w sobie wskazywać na fakt, że w sprawdzanym programie 1-1 jest jakiś błąd, przez występującą w tych warunkach liczbę 8, nie będącą wartością o jakimkolwiek specjalnym znaczeniu dla zadania znalezienia maksimum dowolnych liczb rzeczywistych *a* i *b*.

Problem polega jednak na tym, że w praktyce warunki na poszczególne przypadki testowe nie są znane. Wyznaczenie ich byłoby dzisiaj dla realnych programów praktycznie niemożliwe. Przypadki testowe dla poszczególnych gałązek (ścieżek) są znajdowane albo metodą prób i błędów, albo przy użyciu narzędzi komputerowych, które wewnętrznie wykorzystują wprawdzie warunki na przypadki testowe często analogiczne do rozpatrywanych wyżej, robią to jednak w sposób niejawny: użytkownik otrzymuje tylko konkretne przypadki testowe. Poprawność lub niepoprawność wyników otrzymanych przy zastosowaniu poszczególnych przypadków testowych, jako danych wejściowych dla badanego programu, jest jedyną dostępną informacją co do jego poprawności. W rozpatrywanych bardzo prostych przykładach warunki na przypadki testowe mogą być łatwo znalezione i będą podawane. Umożliwiają one dokładną analizę (obejmującą np. szacowanie wartości prawdopodobieństw) zdolności poszczególnych metod testowania do wykrywania błędów.

„BLACK-BOX-TESTING – testowanie funkcyjne

W testowaniu funkcyjnym przypadki testowe są dobierane na podstawie sformułowania samego zadania. Celem testowania funkcyjnego jest osiągnięcie „całkowitego pokrycia" zadania, przy czym interpretacja tego celu zależy w sposób zasadniczy od charakterystyk(i) zadania, którą(e) przyjęto za podstawę testowania funkcyjnego.

Jeśli przyjętą charakterystyką są *„wszystkie możliwości wynikające z zadania"*, to wybór przypadków testowych dla systematycznego testowania funkcyjnego polega na wyborze jednego przypadku testowego dla każdej zidentyfikowanej możliwości. Możliwości zidentyfikowane dla zadania 1 i 2 prowadzą do następujących wyborów przypadków testowych dla tych zadań.

Dla zadania 1 wybór przypadków testowych jest przy przyjętej charakterystyce zadania jednoznaczny (*f* oznacza testowanie funkcyjne):

Wybór 1-1.1(*f*)
  Przypadek 1-1.1(*f*)I:
    para liczb (*a, b*) taka, że $a > b$
  Przypadek 1-1.1(*f*)II:
    para liczb (*a, b*) taka, że $a < b$

Dla zadania 2 są możliwe dwa różne wybory przypadków testowych:

Wybór 1-2.1(*f*)
  Przypadek 1-2.1(*f*)I:
    trójka liczb (*a, b, c*) taka, że $a > b$ i $a > c$
  Przypadek 1-2.1(*f*)II:
    trójka liczb (*a, b, c*) taka, że $b > a$ i $b > c$
  Przypadek 1-2.1(*f*)III:
    trójka liczb (*a, b, c*) taka, że $c > a$ i $c > b$

Wybór 1-2.2(*f*)
  Przypadek 1-2.2(*f*)I:
    trójka liczb (*a, b, c*) taka, że $a < b < c$
  Przypadek 1-2.2(*f*)II:
    trójka liczb (*a, b, c*) taka, że $a < b < c$
  Przypadek 1-2.2(*f*)III:
    trójka liczb (*a, b, c*) taka, że $b < a < c$
  Przypadek 1-2.2(*f*)IV:
    trójka liczb (*a, b, c*) taka, że $b < c < a$
  Przypadek 1-2.2(*f*)V:
    trójka liczb (*a, b, c*) taka, że $c < a < b$
  Przypadek 1-2.2(*f*)VI:
    trójka liczb (*a, b, c*) taka, że $c < b < a$

Trzeba zauważyć, że zaprezentowane metody testowania nie zawsze można praktycznie zrealizować. Problem znalezienia danych wybierających daną ścieżkę nie jest w ogólności rozstrzygalny. Często np. w przypadku pętli, ścisłe stosowanie zaprezentowanych definicji metod testowania prowadziłoby do ogromnej liczby przypadków testowych, których sprawdzenie byłoby praktycznie niemożliwe. W artykule są rozpatrywane tylko najprostsze przypadki, kiedy omawiane metody testowania można zrealizować, aby podkreślić, że jeśli nawet wtedy metody te mogą być zawodne przy wykrywaniu błędów, to praktycznie w bardziej skomplikowanych przypadkach może być jeszcze gorzej.

* * *

W niniejszej pierwszej części artykułu przedstawiono najbardziej znane metody systematycznego testowania strukturalnego i funkcyjnego. W drugiej części zostanie zbadana zdolność rozpatrywanych metod testowania systematycznego do wykrywania błędów w programach komputerowych.

MEDCOM

CHCESZ ZABEZPIECZYĆ BAZY DANYCH PRZED UTRATĄ INFORMACJI?

CHCESZ ZAPEWNIĆ PRAWIDŁOWĄ PRACĘ

KOMPUTERA * CENTRALI TELEFONICZNEJ * TELEFAKSÓW

W WARUNKACH ZANIKU NAPIĘCIA ZASILAJĄCEGO

kup ZASILACZE BEZPRZERWOWE:

naszej produkcji

UPS – 120 (abonenckie centrale telefoniczne i telefaksy)
UPS – 450
UPS – 850
UPS – 1450

produkcji tajwańskiej:
UPS – 500

Sprzedajemy również:

* AKUMULATORY BEZOBSŁUGOWE, HERMETYCZNE FIRMY PANASONIC do zasilania urządzeń alarmowych, jedyne dopuszczone do używania w pomieszczeniach zamkniętych,
* GENERATORY PRĄDOTWÓRCZE FIRMY ENDRESS O MOCY OD 1 kVA DO 350 kVA.

Twoje problemy w dziedzinie zasilania awaryjnego rozwiążą specjaliści z firmy MEDCOM.

BIURO
Al. Ujazdowskie 26/39
00-478 Warszawa
tel. 28-93-57, teleks 817060

ZAKŁAD PRODUKCYJNO-USŁUGOWY
ul. Bokserska 53
02-690 Warszawa

0/20/90

LESZEK NOWACZYK
Swarzędzkie Fabryki Mebli

Programowanie obiektowe

Programowanie obiektowe można z pewnością zaliczyć do pojęć, które w informatyce zrobiły światową karierę. Chociaż przeważają na jego temat opinie zdecydowanie pozytywne, można się również tu i ówdzie spotkać ze stwierdzeniem, że nie wnosi ono nic istotnego do praktyki programowania i jest bardzo dobrym przykładem na *„mnożenie bytów nad potrzebę"*. Czy jest tak w istocie, czym właściwie jest programowanie obiektowe i dlaczego opłaca się nim zajmować?

Większość pojęć i koncepcji, którymi charakteryzuje się programowanie obiektowe nie jest nowa, jednakże dopiero zastosowanie ich w ramach jednej całości zadecydowało o powstaniu odrębnego stylu. Poniżej przedstawiono krótki opis najważniejszych z tych pojęć.

Współbieżność

W przeciwieństwie do „tradycyjnego" programu, którego wykonanie jest przeważnie procesem sekwencyjnym, program obiektowy wykonywany jest jako system dynamicznie generowanych współbieżnych procesów – obiektów.

Klasy i obiekty

Klasę w podejściu obiektowym można traktować jako rozszerzoną definicję typu. Opisuje ona zarówno strukturę danych, jak też operacje wykonywalne na tychże danych. Definicja klasy jest częścią tekstu źródłowego programu. Na tej podstawie w czasie jego wykonywania są generowane dynamiczne twory – *obiekty*. Każdy obiekt jest reprezentantem, albo *wcieleniem* klasy, która posłużyła jako wzorzec do jego wygenerowania. Teoretycznie w danym momencie istnieć może wiele wcieleń jednej klasy. Jedynym ograniczeniem jest dostępna pamięć komputera. Oczywiście w pamięci mogą jednocześnie znajdować się wcielenia różnych klas.

Dziedziczenie

Definiując klasę, można zadeklarować ją jako *podklasę* innej, już istniejącej. Utworzona w ten sposób struktura odziedziczy wszystkie atrybuty struktury nadrzędnej. Odziedziczone atrybuty mogą być modyfikowane. Można również dodawać nowe atrybuty.

Modułowość

Każdy obiekt stanowi zamkniętą całość. Obejmuje ona lokalną pamięć oraz operacje na niej działające. Dostęp do przyporządkowanej danemu obiektowi pamięci mają tylko operacje w nim zawarte. Operacje te mogą być ujęte w procedury dwojakiego rodzaju. Prywatne, które wywołać może tylko sam obiekt-właściciel oraz udostępnione na zewnątrz.

Message Passing

Ponieważ przyporządkowana danemu obiektowi pamięć jest niedostępna z zewnątrz, jedyną drogą dotarcia do niej jest wywołanie odpowiedniej procedury zawartej w obiekcie, który jest właścicielem tejże pamięci. Odbywa się to pośrednio, na zasadzie przesłania zlecenia do konkretnego adresata. Każda procedura może być uruchomiona tylko w wyniku nadesłania *zlecenia* na jej wykonanie. W przypadku procedur prywatnych, obiekt-nadawca jest równocześnie odbiorcą zlecenia. Przesyłanie *zleceń* (najlepszego, zdaniem autora, w tym kontekście odpowiednika wyrazu *message*) może odbywać się asynchronicznie. Oznacza to, że obiekt-nadawca musi podać adresata zlecenia, natomiast obiekt-odbiorca niekoniecznie musi znać nadawcę. Oczywiście mechanizm ten nie może znaleźć zastosowania w przypadku problemów typu *client/server*. Zlecenia identycznej postaci mogą wywoływać różne efekty, w zależności od tego, do kogo są skierowane. Na przykład, polecenie *RYSUJ* skierowane do obiektu *TRÓJKĄT* spowoduje pojawienie się na monitorze trójkąta, skierowane do obiektu *PSIAK* – wizerunek tego sympatycznego zwierzęcia (oczywiście zakładając, że w obu obiektach istnieją odpowiednie procedury o nazwie *RYSUJ*).

Właściwie już na podstawie podanych wyżej cech można by polecić zorientowane obiektowo języki programowania jako bardzo dobre narzędzia do tworzenia aplikacji. Lokalność danych, ich upakowanie w osobnych modułach wraz z operacjami, uwalnia programistę od zmory nieprzewidzianych efektów ubocznych, które mogą wystąpić zawsze wtedy, gdy większa liczba procesów ma dostęp do globalnych danych. Fakt, że użytkownik może operować obiektami tylko za pośrednictwem przekazanych mu do dyspozycji przez programistę poleceń, decyduje o elastyczności systemu, łatwości dokonywania zmian w szczegółach implementacji, jak również o jego niezawodności. Podobnych zalet można by wymienić jeszcze wiele. Jednakże istnieje też inny powód, dla którego programowanie obiektowe staje się coraz bardziej popularne. Zostanie to poniżej wyjaśnione.

Programowanie obiektowe w systemach wieloprocesorowych

Każdy wygenerowany obiekt może znajdować się w jednym z dwóch stanów: pasywnym i aktywnym. Obiekt przechodzi w stan aktywny, po otrzymaniu skierowanego do niego zlecenia wykonania jakiejś czynności. W tym stanie obiekt może generować inne obiekty, wysyłać zlecenia oraz wykonywać inne czynności, aż do momentu zakończenia obsługi przesłanego zlecenia. W zależności od zachowania się nadawcy, zlecenia dzielą się na następujące trzy grupy:

Grupa 1. Po wysłaniu zlecenia obiekt natychmiast wznawia dalsze działanie. Od adresata zlecenia nie oczekuje się informacji zwrotnej.

Grupa 2. Po wysłaniu zlecenia obiekt przechodzi w stan pasywny, oczekując na zakończenie obsługi zlecenia. Gdy to nastąpi, obiekt ponownie przechodzi w stan aktywny i kontynuuje działanie.

Grupa 3. Po wysłaniu zlecenia obiekt kontynuuje działanie. Osiągnąwszy określony stan sprawdza, czy nadeszła oczekiwana informacja od adresata wysłanego uprzednio zlecenia. Jeśli okaże się, że musi na nią czekać – przechodzi w stan pasywny.

Jeśli z każdym obiektem związemy wirtualny procesor, a następnie odwzorujemy całość w realnie istniejącym systemie wieloprocesorowym, wówczas okaże się, że jeśli w systemie nie występują tylko zlecenia należące do drugiej grupy, to istnieć będą przedziały czasowe, w których większa liczba obiektów będzie znajdować się w stanie aktywnym. Naturalną konsekwencją tego będzie skrócenie czasu wykonywania programu. Oczywiście milcząco zakładamy, że narzut czasowy związany z generowaniem obiektów, wzajemną wymianą informacji i podobnymi czynnościami będzie odpowiednio mały. Założenie to w większości przypadków jest praktycznie spełnione.

Stosunek czasu potrzebnego na wykonanie programu w systemie jednoprocesorowym T_j do czasu potrzebnego w systemie T_w będzie podlegał następującej zależności:

$$1 <= T_j/T_w) <= n$$

gdzie n oznacza liczbę procesorów.
Stosunek ten stanowi miarę efektywności zastosowanego rozwiązania.

dokończenie na s. 20

POZNAJ SIŁĘ NASZYCH SYSTEMÓW INFORMATYCZNYCH!

JUNISOFTEX Sp. z o.o.
44-100 Gliwice ul. Konstytucji 11
Tel. 31-75-10, 31-90-81 do 88 wewn. 250, 272, teleks 036233

Już od 3 lat oferuje kompleksową komputeryzację przedsiębiorstw, a zwłaszcza:

- profesjonalne nowoczesne systemy informatyczne oparte na kompilatorze języka Turbo Pascal 5.5 firmy BORLAND (typu multiusers), pracujące w sieciach NOVELL 2.15 i 3.1 i wykorzystujące interfejs BTRIEV amerykańskiej firmy SoftCraft – najefektywniejszy i najszybszy system obsługi dużych i bardzo dużych baz danych (do 4 miliardów rekordów w pojedynczej bazie danych).

Polecamy szczególnie systemy zintegrowane

- wielodostępny system finansowo-księgowy wg trójstopniowej analityki w podstawowym układzie: xxx-xxxx-xxxx-xxxxxx z możliwością graficznej prezentacji wyników,
- wielodostępny system gospodarki materiałowej wykorzystujący w strukturze indeksowej SWW i KTM z automatycznym wyliczaniem cyfry kontrolnej (długość indeksu materiałowego 13 + 3 cyfry),
- wielodostępny system gospodarki wyrobami gotowymi wykorzystujący w strukturze indeksowej SWW i KTM,
- wielodostępny system gospodarki przedmiotami nietrwałymi wykorzystujący w strukturze indeksowej SWW i KTM,
- wielodostępny system środków trwałych wykorzystujący klasyfikację środków trwałych GUS,
- wielodostępny system fakturowania sprzedaży,
- wielodostępny system kadrowo-płacowy.

Wszystkie oferowane systemy są zintegrowane ze sobą zgodnie z zasadami rachunkowości księgowej przedstawionymi w miesięczniku RACHUNKOWOŚĆ nr 9, 1990.

Oferujemy również:

- sprzęt komputerowy (sprawdzony w eksploatacji) firm:
  – ALR (Advanced Logic Research) i Storage Dimensions z USA,
  – AZTECH, WEARNES TECHNOLOGY, NAC i inne z Singapuru,
- dostawę i instalację licencjonowanej sieci Novell 2.15 i 3.1,
- usługi w zakresie opracowywania zakładowych indeksów materiałowych wg klasyfikacji GUS (SWW i KTM).

Organizujemy:

- szkolenia i prezentację systemów u naszych kontrahentów.

Gwarantujemy:

- najwyższą jakość naszych systemów i usług,
- najszybsze w Polsce wdrożenia systemów (od 1 do 3 miesięcy).

W przypadku niedotrzymania przez nas terminów zwracamy 25% poniesionych nakładów za każdy miesiąc zwłoki. Dla klientów zamawiających kompleksowe usługi – znaczne zniżki cenowe.

Poszukujemy niezawodnych

- pośredników i dealer'ów naszych systemów informatycznych (oferujemy bardzo atrakcyjne warunki współpracy),
- dostawców i pośredników dobrego i taniego sprzętu komputerowego.

Z siły naszych systemów skorzystały dotąd następujące firmy:

Huta ŁAZISKA w Łaziskach Górnych	tel. 24-14-06
Walcownie Metali w Czechowicach-Dziedzicach	tel. 52-35-1
Centrala Zaopatrzenia Górnictwa w Bytomiu	tel. 81-92-91
Mikołowska Fabryka Transformatorów w Mikołowie	tel. 26-25-41
BPiKD PREFAMET-ZREMB w Gliwicach	tel. 38-20-51
BPiKD METALCHEM w Gliwicach	tel. 31-56-33
Instytut Informatyki Teoretycznej i Stosowanej PAN w Gliwicach	tel. 31-71-50
Instytut Spawalnictwa w Gliwicach	tel 31-00-11
PP ENERGOROZRUCH w Gliwicach	tel. 37-66-25
Fabryka Palenisk Mechanicznych w Mikołowie	tel. 26-20-02
Zakłady Mechaniczne Górnictwa i Energetyki WIROMET w Mikołowie	tel. 26-23-46
Przędzalnia Bawełny PRZYJAŹŃ w Zawierciu	tel. 22-39
PRT TERMOIZOLACJA w Zabrzu	tel. 71-40-21
Przedsiębiorstwo Usług Informatycznych ETOM w Mikołowie	tel. 57-10-50
Zakład Informatyki i Techniki Komputerowej MEGA w Gliwicach	tel. 31-52-42
Mikołowska Fabryka Maszyn Górniczych MIFAMA w Mikołowie	tel. 26-22-66
Zakłady Produkcji Urządzeń Mechanicznych ELWO w Pszczynie	tel. 30-61
Zakłady Sprzętu Sieciowego i Elektroinstalacyjnego POLAM w Katowicach	tel. 52-80-31
Urząd Miasta w Mikołowie	tel. 26-23-90
Zakłady Elektromaszynowe CELMA w Cieszynie	tel. 21-58-1
PSS SPOŁEM w Zawierciu	tel. 21-46-17

Chcesz mieć pełną gwarancję i niezawodne efekty komputeryzacji swojego przedsiębiorstwa – współpracuj z firmą JUNISOFTEX!

0/1/91

TOMASZ BERUS
BOHDAN MACUKOW
Instytut Matematyki
Politechnika Warszawska

Od sieci neuronowych do neurokomputerów

Wyobraźmy sobie komputer, który rozwiązując pewien problem sam się uczy. Wprowadzamy do niego informacje o zadaniu, dane wejściowe oraz wybrane przykłady wraz z poprawnymi rozwiązaniami. Albo inaczej, maszyna sama pobiera informacje z otaczającej ją rzeczywistości za pomocą swych urządzeń zewnętrznych. Następnie komputer analizuje wprowadzone informacje i ucząc się na swoich błędach w końcu osiąga taki stan, w którym postawiony problem może być rozwiązany. W działalności tego typu można zauważyć wiele podobieństwa do działania człowieka. I tutaj pojawia się pytanie: czy potrafimy skonstruować urządzenie techniczne o podobnych własnościach? Badania ostatnich lat sugerują odpowiedź twierdzącą, a sieci neuropodobne wydają się być drogą do tego celu.

Do zaprojektowania systemów tego rodzaju wykorzystano modele sieci neuronowych. Podstawą tych modeli są ogólne zasady działania mózgu ludzkiego oraz wiadomości o budowie i zasadach działania jego pojedynczych elementów – neuronów. Wykorzystano także znane metody organizacji tych prostych elementów w jednostki funkcjonalne – w złożone struktury sieciowe. Wiadomo, że takie struktury zdolne są do wykonywania pewnych działań, do rozwiązywania problemów podobnych do tych, jakie rozwiązuje mózg człowieka. Tak powstała idea komputera zupełnie odmiennego od dotychczas istniejących, komputera wykorzystującego całkowicie różne zasady działania i zbudowanego z innych elementów – idea *neurokomputera*.

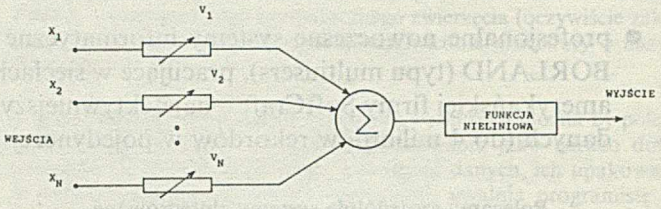
Neurokomputery można określić zarówno jako komputery, których budowa jest podobna do biologicznej struktury mózgu ludzkiego, jak i programy komputerowe, które przeprowadzają obliczenia w sposób zbliżony do zasad działania mózgu. Oznacza to, że zarówno w pierwszym jak i w drugim podejściu symulują lub emulują one rzeczywiste sieci neuronowe. W licznych publikacjach neurokomputery występują pod różnorodnymi nazwami. Najbardziej popularne z nich to: komputery neuronowe, modele sieci neuronowych, sztuczne systemy neuronowe (nerwowe), elektroniczne sieci neuronowe, równoległe sieci skojarzeniowe, równoległe procesory czy komputery szóstej generacji.

Dziedzina neurokomputerów, albo inaczaj – obliczeń neurokomputerowych (ang. *neurocomputing*) jest często określana jako dziedzina badań inżynierskich zajmująca się nieprogramowymi, adaptacyjnymi systemami przetwarzania informacji. Mówiąc precyzyjniej, jest to dziedzina zajmująca się wpływem struktur połączeń sieciowych, schematów połączeń i sposobów współdziałania poszczególnych elementów sieci na zasady działania sieci jako całości.

W niniejszym artykule postaramy się omówić podstawowe cechy układów typu sieciowego zbudowanych z elementów neuropodobnych, a także możliwości różnych typów neurokomputerów w porównaniu z konwencjonalnymi komputerami cyfrowymi.

Co to są neurony i sieci neuronowe?

Sztuczne sieci neuronowe są abstrakcyjnymi modelami rzeczywistego układu nerwowego. Sieć zbudowana jest z elementów przetwarzających zwanych neuronami, lub poprawniej – elementami neuropodobnymi. Pojedynczy element jest określony przez położenie w sieci, stan wewnętrzny, układ połączeń z innymi elementami, stan pobudzenia oraz charakterystykę opisaną nieliniową i progową funkcją. Jest to element sumujący wszystkie sygnały pobudzające i hamujące, które do niego dochodzą (rys. 1).



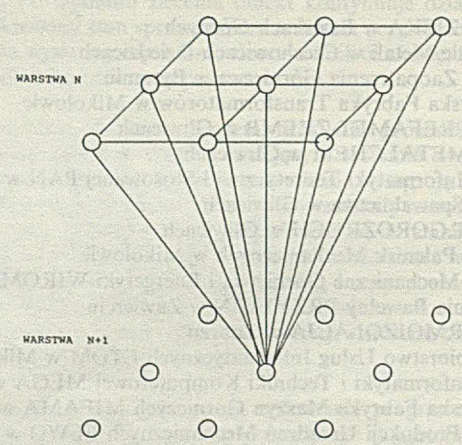
Rys. 1. Model neuronu

W przypadku gdy przeważają sygnały pobudzające, tzn. gdy przekroczona zostanie pewna wartość progowa θ, neuron wysyła sygnały do wszystkich swoich wyjść.

$$y = f\left\{\sum_{i=1}^{n} x_i v_i - \theta\right\}$$

gdzie x_i są sygnałami wejściowymi pochodzącymi od innych elementów sieci, a v_i wagami (współczynnikami) sprzężeń. To z kolei powoduje pobudzenie (lub hamowanie) neuronów powiązanych z nim tymi połączeniami.

Neurony są ułożone w warstwy, tworząc uporządkowane, najczęściej wielowarstwowe struktury (rys. 2). Neuron może być połączony z wieloma innymi elementami danej warstwy, włącznie z samym sobą. Mogą to być połączenia w ramach danej warstwy, międzywarstwowe, w przód lub w tył. Siła (waga) połączenia może być dodatnia (mówimy wtedy o połączeniu pobudzającym), ujemna (połączenie hamujące) lub zerowa – brak połączenia. Waga połączenia może być ustalona lub zmienna w czasie.



Rys. 2. Fragment sieci warstwowej

Oczywiście opis powyższy stanowi ogromne uproszczenie w porównaniu do faktycznej struktury mózgu ludzkiego. W rzeczywistości w układzie nerwowym człowieka jest około 5×10^{10} neuronów, które oddziałują na siebie poprzez około 10^{14} połączeń i ich działanie jest znacznie bardziej skomplikowane niż działanie opisanych powyżej struktur.

Dlaczego wykorzystujemy sieci neuronowe?

Mózg człowieka ciągle jest najpotężniejszym z istniejących obecnie urządzeń liczących do celów przetwarzania informacji w czasie rzeczywistym.

Biologiczne sieci nerwowe mogą przeprowadzać bardzo skomplikowane obliczenia i działania, aczkolwiek jest rzeczą oczywistą, że w np. rachunkach, w działaniach arytmetycznych – mózg ludzki nie jest tak dobrym urządzeniem (tak szybkim, wydajnym i dokładnym) jak komputer. Ale z drugiej strony, gdy mamy do czynienia z zadaniami takimi jak rozpoznawanie, klasyfikacja czy skojarzenia – mózg może pokonać nawet najszybszy superkomputer.

Jest także ogromna różnica między metodami działania sieci neuronowej układu nerwowego, a sposobem, w jaki komputer rozwiązuje problem. Przykładowo, gdy mamy obliczyć 4×5, to przecież nie wykonujemy tego mnożenia tylko rozpoznajemy problem, a następnie przywołujemy z pamięci właściwą, skojarzoną odpowiedź wynikającą z faktu, że kiedyś w dzieciństwie uczyliśmy się na pamięć tabliczki mnożenia.

Jedną z najciekawszych własności, najbardziej różniącą świat sieci nerwowych od świata komputerów jest zdolność do tolerowania i poprawiania błędów. Mózg zdolny jest do rekonstrukcji, odtworzenia sygnału na podstawie informacji częściowej i często dodatkowo obarczonej błędami. Mózg ma niesamowitą wręcz zdolność do rekonstrukcji informacji na podstawie małego jej fragmentu. Przykładowo, gdy patrzymy na bardzo mały i zniszczony fragment zdjęcia znajomej osoby, zazwyczaj nie mamy trudności z dokonaniem właściwego rozpoznania.

Analizując metody przetwarzania i selekcji informacji oraz sposoby podejmowania decyzji w systemie nerwowym łatwo dojść do wniosku, że jest on przykładem rozwiązania wielu problemów, z którymi do dzisiaj boryka się informatyka, teoria przetwarzania informacji czy teoria optymalizacji.

Jednocześnie badania nad modelami sieci neuronowych mogą w ogromnym stopniu ułatwić pracę biologom, neurofizjologom itp. Prawidłowa interpretacja i analiza wyników badań neurofizjologicznych czy elektrofizjologicznych napotyka stale na bardzo wielkie trudności, ponieważ są one wynikiem pomiarów procesów dynamicznych w złożonych sieciach o nieznanej dokładnie strukturze i bliżej nieokreślonych parametrach elementów. Z jednej strony wyniki badań dostarczają ogromnej liczby danych o pracy poszczególnych fragmentów systemu nerwowego, a z drugiej, napotykamy na ogromne luki i niejasności, a często nawet sprzeczności wyników.

Niestety nadal jesteśmy zmuszeni do ograniczenia swoich zainteresowań do poszczególnych fragmentów, bądź funkcji systemu nerwowego, wypełniając liczne luki mniej lub bardziej prawdopodobnymi hipotezami. Wykorzystanie metod matematycznych i nowoczesnych urządzeń technicznych, a zwłaszcza elektronicznych, może w istotny sposób przyczynić się do przełamania tych trudności.

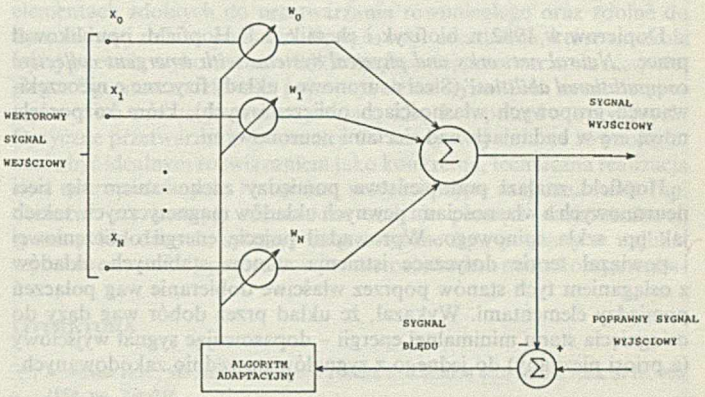
Rys historyczny

Pierwsze spostrzeżenia na temat pobudzania tkanek i towarzyszących im zjawisk elektrycznych pojawiają się w XVIII i XIX wieku. Odkrycie, że przekazywanie bodźców w systemie nerwowym jest związane z powstawaniem w komórce nerwowej impulsów elektrycznych o stałej amplitudzie – zapoczątkowało budowę modeli neuronu.

Niewątpliwie punktem początkowym badań w tej dziedzinie była słynna praca McCullocha i Pittsa (neurofizjolog i matematyk) z 1943 roku *„A logical calculus of the ideas immanent in nervous activity"* (Opis logiczny idei dotyczących aktywności nerwowej) dotycząca modelowania zjawisk w systemie nerwowym i zasad konstruowania sieci złożonych z prostych elementów. Przedstawili oni m.in. prosty model matematyczny neuronu jako tzw. *elementu logicznego,* działającego wg zasady *wszystko albo nic,* to znaczy element ten albo znajduje się w stanie spoczynku (stan zero), albo – gdy pubudzenie przekroczy pewien próg – w stanie pobudzenia (stan 1). Z tego typu elementów rozpoczęto budowę sieci, tzw. *sieci logicznych.* Późniejsze badania dotyczące własności neuronu, a szczególnie prace Ecclesa, Hodgkina i Huxleya (za które w 1963 r. otrzymali nagrodę Nobla) umożliwiły bardziej dokładny opis własności neuronu, który przecież nie może być traktowany jedynie jako element cyfrowy reagujący wg zasady *„wszystko albo nic"*, ale także jako tzw. element analogowy, tzn. że zachodzące w nim procesy zależą w sposób ciągły od pobudzenia.

Na przełomie lat pięćdziesiątych i sześćdziesiątych pojawiła się duża liczba prac, w których przedstawiano różnorodne technicznie, najczęściej elektroniczne, modele neuronu. Do najbardziej znanych należą modele Harmona, Taylora, Kupfmillera i Jenika.

Również tacy naukowcy jak Kleene, J. von Neumann, Culbertson czy Moore prowadzili prace nad modelami oraz sposobami ich opisu matematycznego.

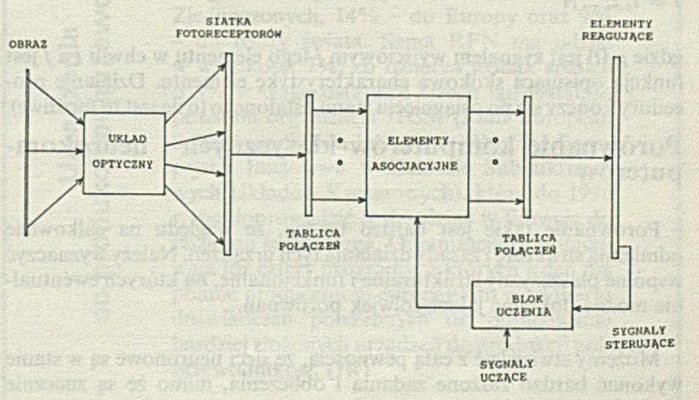


Rys. 3. ADELINE – adaptacyjny neuron liniowy

W 1960 roku Bernard Widrow i Marcian Hoff zaproponowali model tzw. *adaptacyjnego neuronu liniowego* (*ADAptive LInear NEuron* – ADALINE) – rys. 3, który faktycznie był jednym z pierwszych neurokomputerów. Urządzenie wykorzystywało regułę uczenia, zwaną dzisiaj regułą *Widrowa-Hoffa* i często stosowaną w neurokomputerach dnia dzisiejszego. Zasada działania była prosta. Różnica pomiędzy sygnałem wyjściowym urządzenia a sygnałem, jaki pragnęliśmy na tym wyjściu uzyskać, jest poprzez pętle sprzężenia zwrotnego wykorzystywana do modyfikacji wag połączeń. W ten sposób kolejne wyjście jest bardziej „poprawne" i procedura trwa aż do uzyskania właściwego sygnału wyjściowego. Ich urządzenie było zdolne do rozpoznawania prostych obrazów, a nawet mogło opracowywać prognozy pogody.

W roku 1962 informatyk, Frank Rosenblatt, w swojej książce *„Principles of Neurodynamics"* opisał teorię i ideę perceptronu (rys. 4). Jest to urządzenie zbudowane na bazie warstwy elementów neuropodobnych. Całość może być strukturą jedno- lub wielowarstwową. Zgodnie z pierwotną ideą Rosenblatta, warstwa pierwsza, wejściowa, była odpowiednikiem tablicy receptorów znajdujących się w oku. Warstwa końcowa zbudowana była z elementów, z których każdy reagował tylko na jeden obraz wejściowy. Elementy warst pośrednich nosiły nazwę elementów skojarzeniowych (asocjacyjnych).

Poszczególne warstwy przekazują sygnał w sposób zawężający tzn. sygnały wyjściowe określonej grupy elementów danej warstwy stanowią



Rys. 4. Perceptron Rosenblatta

sygnał wejściowy dla jednego z elementów warstwy następnej. Prace i wyniki Rosenblatta były bardzo obiecujące. Wydawało się, że jest to odkrycie, które pozwoli na studiowanie działalności mózgu – co więcej – pozwoli nawet na zbudowanie jego repliki. Rosenblatt twierdził, że przyszłość badań to tylko urządzenia typu perceptron, a nie metody i idee sztucznej inteligencji.

W 1969 r. Marvin Minsky i Seymour Papert rozprawą matematyczną *„An introduction on computational geometry"* praktycznie zniszczyli ideę perceptronu. Odpowiednio dobranymi przykładami pokazali jego niedoskonałość i ograniczenia, twierdząc ponadto, że te wady spotęgują się dla sieci bardziej złożonych. Stało się to także bezpośrednią przyczyną przerwania badań nad sieciami neuropodobnymi na kilkanaście następnych lat.

Dopierow w 1982 r. biofizyk i chemik, J.J. Hopfield, opublikował pracę *„Naural networks and physical systems with emergent collective computational abilities"* (Sieci neuronowe i układy fizyczne o nieoczekiwanych grupowych własnościach obliczeniowych), która rozpoczęła nową erę w badaniach nad sieciami neuronowymi.

Hopfield znalazł podobieństwa pomiędzy zachowaniem się sieci neuronowych a własnościami pewnych układów magnetycznych, takich jak np. szkła spinowego. Wprowadził pojęcie energii obliczeniowej i powiązał teorie dotyczące istnienia stanów stabilnych układów z osiąganiem tych stanów poprzez właściwe dobieranie wag połączeń pomiędzy elementami. Wykazał, że układ przez dobór wag dąży do osiągnięcia stanu minimalnej energii – dopasowując sygnał wyjściowy (a priori nieznany) do jednego z sygnałów uprzednio zakodowanych.

Na rysunku 5 pokazany jest bardzo prosty, dwuwarstwowy, o czterech elementach w każdej warstwie, model sieci Hopfielda. Sieć zbudowana jest z identycznych nieliniowych elementów (o charakterystyce skokowej) i binarnych sygnałach wejściowych oraz wyjściowych (0 i + 1 albo − 1 i + 1). Sygnał wyjściowy każdego z elementów warstwy pierwszej jest podawany na wejścia elementów warstwy drugiej z odpowiednią wagą t_{ij}. Wagi te są ustalane na etapie uczenia i zależą od zapamiętywanych wzorców.

$$t_{ij} = \begin{cases} \sum_{s=1}^{M} x_j^s x_i^s & \text{dla } i \neq j \\ 0 & \text{dla } i = j \end{cases}$$

$$i,j = 1,2,...,N$$

gdzie t_{ij} jest wagą połączenia wyjścia i-tego elementu warstwy pierwszej z wejściem j-tego elementu warstwy drugiej ($t_{ij} = t_{ji}$), $x_i^s (= -1, +1)$ jest i-tą składową s-tego wzorca ($s = 1,2,...,M$).

Po etapie uczenia na elementy wejściowe sieci w chwili $\tau = 0$ podany jest nieznany sygnał $X = [x_1,...,x_N]$ i sieć ma iteracyjnie poprzez pętle sprzężenia zwrotnego „dopasować" sygnał wyjściowy do sygnału wejściowego zgodnie z procedurą

$$\mu_j(0) = x_j$$

$$\mu_j(t+1) = f\left[\sum_{i=1}^{N} t_{ij}\mu_i(t)\right]$$

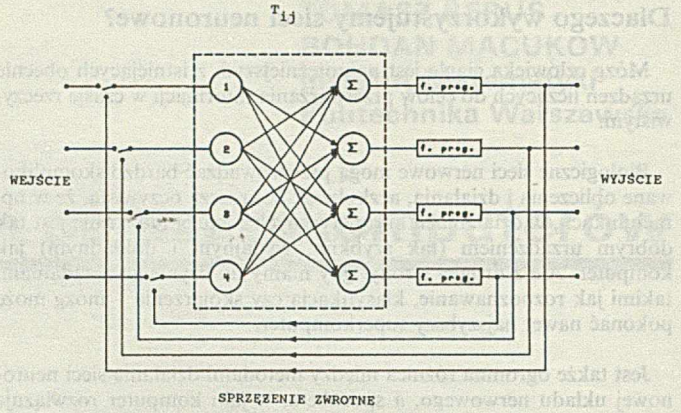
$$J = 1,2,...,N$$

gdzie $\mu_j(t)$ jest sygnałem wyjściowym j-tego elementu w chwili t a f jest funkcją opisującą skokową charakterystykę elementu. Działanie procedury kończy się po osiągnięciu stanu ustalonego (o ile jest to możliwe).

Porównanie komputerów klasycznych i neurokomputerów

Porównanie takie jest bardzo trudne, ze względu na całkowicie odmienną strukturę i zasady działania tych urządzeń. Należy wyznaczyć wspólne płaszczyzny strukturalne i funkcjonalne, na których ewentualnie można dokonać jakichkolwiek porównań.

Możemy stwierdzić z całą pewnością, że sieci neuronowe są w stanie wykonać bardzo złożone zadania i obliczenia, mimo że są znacznie wolniejsze niż układy elektroniczne. Pojedynczy neuron może przekazywać około tysiąc impulsów na sekundę, podczas gdy elektroniczny układ scalony – miliard i więcej. Z drugiej strony, równoległa budowa i duża liczba połączeń w sieci prowadzi do bardzo zaawansowanego przetwarzania równoległego. Neuron ma ok. 10 tys. połączeń, podczas gdy element elektroniczny nie więcej niż 100. Wydaje się być oczywiste, że wielka moc obliczeniowa mózgu jest właśnie związana ze wspomnianą ogromną liczbą połączeń, które pozwalają na przetwarzanie równolegle, co kompensuje małą szybkość przetwarzania. W tabelach 1 i 2 przedstawiono porównanie wybranych cech funkcjonalnych, strukturalnych i rozwiązań sprzętowych konwencjonalnych komputerów cyfrowych oraz neurokomputerów.



Rys. 5. Model sieci Hopfielda

Tabela 1. Porównanie cech funkcjonalnych

Cecha	Neurokomputery	Komputery konwencjonalne
Czułość na sprzężenie	bardzo dobra	minimalna
Pamięć	duża gęstość, rozłożona, skojarzeniowa	mała gęstość zlokalizowana,
Przeszukiwanie zbiorów danych	szybkie, zbliżone dopasowanie	wolne, dokładne dopasowanie
Obliczenia matematyczne i algorytmiczne	słabe	bardzo dobre
Zdolności heurystyczne	bardzo dobre	słabe
Rozpoznawanie obrazów	szybkie	wolne
Rozpoznawanie obrazów niekompletnych	bardzo dobre	słabe

Tabela 2. Porównanie cech strukturalnych i rozwiązań sprzętowych

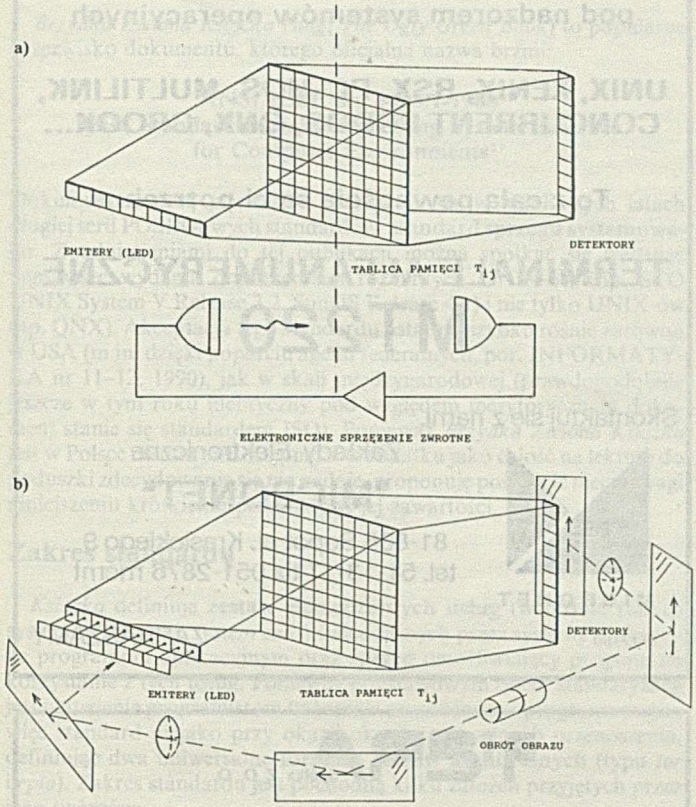
Cecha	Neurokomputery	Komputery konwencjonalne
Reprezentacja danych	quasi-analogowa	cyfrowa
Połączenia elementów przetwarzających	około 10, dynamicznie zmienne przez samoprogramowanie	około 3, nie zmieniające się
Sposób przetwarzania	równolegle	szeregowo
Lokalizacja pamięci, logiki i sterowania	nielokalna, rozłożona w połączeniach	umiejscowiona w elementach przetwarzających
Typ elementów przetwarzających	nieliniowe, mogą być nieprogowe,	liniowe z progiem,
Struktura budowy	elementy ułożone równolegle	elementy ułożone szeregowo

Koncepcje budowy neurokomputerów

Ogólną cechą charakteryzującą wszystkie koncepcje budowy neurokomputera jest to, że urządzenie to będzie działało nie na zasadzie algebry Boole'a, tak popularnej w konwencjonalnych komputerach, ale na zasadzie np. algebry macierzowej.

Głównymi cechami charakterystycznymi neurokomputerów są:
- możliwość pracy równoległej,
- skojarzeniowy dostęp do pamięci,
- zapis informacji we wszystkich komórkach pamięci (a mówiąc dokładniej – w strukturze połączeń), co między innymi daje możliwość pracy komputera nawet w przypadku, gdy pewien procent elementów pamięci jest uszkodzony.

Neurokomputer wymaga ogromnej liczby połączeń między elementami pamięci. Sieć połączeń w neurokomputerze jest trójwymiarowa w przeciwieństwie do stosowanych obecnie sieci dwuwymiarowych. Ta ogromna liczba połączeń powoduje niebezpieczeństwo oddziaływań między połączeniami, które mogą uniemożliwić prawidłowe działanie całego systemu. Sugeruje się zastosowanie w takim komputerze połączeń optycznych, które ze względu na szybkość i niewrażliwość na zaburzenia wydają się być idealne dla tego typu zastosowań.



Rys. 6. Optyczna realizacja modelu Hopfielda ze sprzężeniem zwrotnym: a) elektronicznym, b) optycznym

Jedną z możliwych realizacji optycznych modelu Hopfielda przedstawiono na rys. 6a i 6b. Sygnałem wejściowym jest układ N świecących diod (LED) reprezentujących N elementów w stanie 0 albo 1. Macierz pamięci T o elementach $-1,0$ i 1 jest realizowana w formie przeźrocza. Układ optyczny (niewidoczny na rysunku dla większej jego czytelności) dokonuje dobrze znanej operacji mnożenia wektora przez macierz. Światło pochodzące z diod świecących jest przez anamorficzny układ optyczny kierowane na macierz T, a po jej przejściu jest ogniskowane przez następny układ optyczny, rejestrowane przez wektor diod i ponownie kierowane poprzez pętle sprzężenia zwrotnego na wejście. Układ nieliniowego sprzężenia zwrotnego (ze wzmocnieniem w celu kompensacji strat) można realizować w dwojaki sposób: elektronicznie (rys. 6a) oraz optycznie (rys. 6b).

W jaki sposób tworzyć komputery równoległe? Najprostszą metodą jest połączenie kilku procesorów w sieć. W tym przypadku, każdy element sieci komputerowej – pojedynczy mikroprocesor, jest elementem o skomplikowanej budowie. Kierunek rozwoju komputerów równoległych jest taki, iż wraz ze wzrostem liczby procesorów w sieci maleje ich stopień złożoności, a wzrasta liczba połączeń między nimi. W przypadku neurokomputerów mówimy o tzw. *architekturze drobnoziarnistej* lub inaczej *rozsianej*. W tym przypadku pojedynczy procesor jest to element neuropodobny, przyjmujący wartości 0 lub 1. Informacja zapamiętywana jest we wszystkich elementach równocześnie, analogicznie do informacji kodowanej w procesach holograficznych.

Przyszłość neurokomputerów

Od kilku lat na rynku jest dostępnych wiele różnorodnych programów symulujących neurokomputery przy pomocy typowych mikrokomputerów. Są to głównie programy demonstracyjne albo programy pozwalające na rozwiązywanie stosunkowo prostych problemów. Poza nielicznymi próbami, jest to zazwyczaj symulacja przy wykorzystaniu typowego, konwencjonalnego sprzętu i z tego powodu wymagane są duże pamięci i duże szybkości obliczeniowe. To z kolei bardzo ogranicza ich możliwości i użyteczność. Dlatego też zaczyna stosować się specjalne pakiety typu koprocesor, gdzie szybkość przetwarzania może być znacznie większa niż w urządzeniu typowym.

W Stanach Zjednoczonych w ostatnich latach pojawiło się wiele firm oferujących takie oprogramowanie albo zestawy sprzętowo-programowe. Należą do nich m.in. Hecht-Nielsen Neuro-Computer, SAIC Synaptics, NeuralWare Inc. czy Nestor. Wydaje się, że w ciągu najbliższych kilku lat powinny pojawić się już pierwsze „prawdziwe" neurokomputery zbudowane na innych niż dzisiaj zasadach, oparte na elementach zdolnych do przetwarzania równoległego oraz zdolne do dynamicznej zmiany połączeń. Będą to więc urządzenia, które znacznie lepiej pozwolą wykorzystać wielkie możliwości przetwarzania równoległego. Ogromne możliwości pojawiają się na styku teorii neurokomputerów z ich realizacją wykorzystującą ideę komputerów optycznych. Optyczne przetwarzanie informacji (w samej swej naturze równoległe) może być idealnym rozwiązaniem jako konkretna, techniczna realizacja sieci neuropodobnych. Natomiast szeroko pojęte rozpoznawanie (np. obrazów czy mowy), różnorodne rodzaje pamięci skojarzeniowej, systemy baz danych, systemy diagnostyczne itp., powinny w najbliższym okresie stać się podstawowym zastosowaniem neurokomputerów.

LITERATURA

[1] Arsenault H.H., Macukow B.: Beyond Pattern Recognition with Neural Nets. SPIE, 960, 1988, pp. 206–216
[2] Arsenault H.H., Macukow B.: Neural Network Model for fast Learning and Retrieval. Opt. Engin. 28, 1989, pp. 506–512
[3] Arsenault H.H., Macukow B.: Neural Networks for Logic Operations. SPIE, 1134, 1989, pp. 40–42
[4] Berus T., Macukow B.: On Some Modification of Hopfield's Content – Addressable Memory. SPIE, 1319, 1990, p. 151
[5] Farhat N.H., Peak E., Prata A., Psaltis D.: Optical Implementation of the Hopfield Model. Appl. Opt. 24, 1985, pp. 1469–1475
[6] Hecht-Nielsen R.: Neurocomputing: Picking the Human Brain. IEEE Spectrum, March 1988, pp.36–41
[7] Hopfield J.J.: Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proc. Natl. Acad. Sci. USA, 79, 1982, pp. 2554–2558
[8] Macukow B.: Neurokomputery. Postępy Fizyki (w druku)
[9] McCulloch W., Pitts W.: A Logical Calculus of Ideas immanent in Nervous Activity. Bull. Math. Bioph., 5, 1943, p. 115
[10] Minsky M., Papert S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969
[11] Rosenblatt F.: Principles of Neurodynamics. Spartan Books, New York, 1962
[12] Widrow B.: Generalization and Information Storage in Networks of Adaline „Neurons", w: Self Organizing systems 1962 pod red. M.C. Yovitz, G.T. Jacobi i G. Goldstein, Spartan Books, Washington DC, 1962, pp. 435–461

Urządzenia do produkcji półprzewodników

Jak wynika z danych opublikowanych przez Stowarzyszenie Niemieckich Wytwórców Maszyn i Zakładów Przemysłowych, około 90% urządzeń wykorzystywanych do produkcji półprzewodników w Europie pochodzi z importu. Obecnie 45% światowego rynku tych urządzeń należy do Japonii, 32% – do Stanów Zjednoczonych, 14% – do Europy oraz 9% – do reszty świata. Sama RFN ma udział w wysokości 7%, co oznacza jednak spadek o 3% w ciągu ostatnich trzech lat. Ostatnio powstała organizacja JESSI (Joint European Submicron Silicon Initiative – Wspólna Europejska Inicjatywa w Zakresie Submikronowych Układów Krzemowych), która do 1996 r. ma doprowadzić tę dziedzinę w Europie do poziomu światowego. Organizacja ta opracowała już cztery programy, z których jeden ma pomóc producentom europejskim w zdobyciu doświadczeń potrzebnych do opracowania bardziej złożonych urządzeń do produkcji półprzewodników. (JR)

ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH

MiCOMP

40–045 Katowice, ul. Astrów 7
telefon 513-086, telefaks 518-628,
teleks 315687

Systemy teletransmisji danych
ICL/ODRA

Integracja systemów ICL i ODRA z mikrokomputerami standardu PC i systemami NOVELL, UNIX SCO

procesor sieci rozległej WAN
MiCOMP 8075
(emulacja ICL 7503)

most NOVELL – ICL/ODRA
NetCROSS-FXBM
program teletransmisji danych
MiCROS-FXBM (PC, rozproszone bazy danych, wersja sieciowa – NOVELL),

skaner **MPXSCAN-8007,**

adaptery, modemy, testery,
emulatory terminali

karty BSC do połączeń komputerów:
PC/XT/AT z PC/XT/AT
PC/XT/AT z **IBM**/370, R32, R34
z emulatorami IBM 3270/3780
PC/XT/AT z **ICL**

karty kompatybilne z **IRMA** do połączeń
PC/XT/AT z IBM 370
lokalnie przez IBM 3274

karty **SNA/SDLC**

konwertery o szybkościach do 19200 bps
do zdalnego przyłączania terminali

przystawki teleksowe współpracujące
z mikrokomputerami PC/XT/AT

Instalacje w największych systemach
na terenie całego kraju!

0/23/90

Posiadasz maszyny

typu IBM, serii SM, PDP-11, MERA400...,

pracujące

pod nadzorem systemów operacyjnych

UNIX, XENIX, RSX, PC-MOS, MULTILINK, CONCURRENT PCDOS, QNX, CROOK...

To z całą pewnością są ci potrzebne

TERMINALE ALFANUMERYCZNE
MT220

Skontaktuj się z nami.

MICRONET

Zakłady Elektroniczne
"MICRONET"
81-836 Sopot ul. Krasickiego 9
tel. 51-13-17 tlx 051-2876 mcrnt

0/17/90

ZAPRASZAMY

TERA Spółka Z O. O.

Przedsiębiorstwo Popierania Postępu TERA Spółka z o.o. uprzejmie informuje, że posiadając kilkuletnie doświadczenie w instalacji systemów wspomagających zarządzanie

o f e r u j e :

- opracowanie projektów systemów,
- optymalny kosztowo i rozwojowo dobór sprzętu i oprogramowania,
- instalację systemu u klienta,
- bezpłatnie przez rok konserwację oprogramowania oraz serwis sprzętu wraz z doradztwem techniczno-eksploatacyjnym,
- szybką dostawę uzupełnień konfiguracji lub sprzętu mikrokomputerowego niezależnego od systemu, w tym mikrokomputerów ALR firmy Wearnes Technology.

Wszelkie dodatkowe informacje uzyskają Państwo codziennie oprócz niedziel w Biurze Handlowym
40–025 Katowice, ul. 27 Stycznia 31/17
tel. (faks): 51-61-65, teleks: 315448 tera pl

PAMIĘTAJ! Instalacje XENIX/NOVELL/PC MOS 386 oraz serwis to nasza specjalność – ZAPRASZAMY, ponieważ czterech lat doświadczeń nigdzie nie kupisz.

0/19/90

/standardy/POSIX.1

Brzydka Zielona Książka

Brzydka Zielona Książka (ang. The Ugly Green Book) to popularne przezwisko dokumentu, którego oficjalna nazwa brzmi:

ANSI/IEEE Std 1003.1-1988
IEEE Standard Portable Operating System Interface for Computer Environments[1]

Dokument zawiera pierwszy z czekającej nas w najbliższych latach długiej serii POSIX-owych standardów: **standard sprzęgu systemowego**. Z odniesieniami do tej publikacji można spotkać się wertując najnowsze wydania dokumentacji rozmaitych UNIX-ów (np. SCO UNIX System V Release 3.2, SunOS Release 4.1) i nie tylko UNIX-ów (np. QNX). Akceptacja tego standardu ostatnio szybko rośnie zarówno w USA (m.in. dzięki poparciu agend federalnych, por. INFORMATYKA nr 11–12, 1990), jak w skali międzynarodowej (prawdopodobnie jeszcze w tym roku identyczny pod względem merytorycznym dokument stanie się standardem ISO). Ponieważ *Brzydka Zielona Książka* jest w Polsce niemal niedostępna, a w dodatku jako całość na lekturę do poduszki zdecydowanie się nie nadaje, proponuję poświęcić nieco uwagi niniejszemu krótkiemu przeglądowi jej zawartości.

Zakres standardu

Książka definiuje **zestaw elementarnych usług** (w formie funkcji systemowych, ang. *system calls*) świadczonych przez systemy operacyjne programom aplikacyjnym oraz **sprzęg** umożliwiający programom korzystanie z tych usług. Ponieważ podstawowym celem standaryzacji jest ułatwienie programistom tworzenia przenośnego oprogramowania, więc standard niejako przy okazji określa sam sposób przenoszenia, definiując dwa uniwersalne **formaty plików archiwalnych** (typu *tar* i *cpio*). Zakres standardu jest pochodną kilku założeń przyjętych przez jego twórców:

● **Przenośność aplikacji.** Standard definiuje minimalny sprzęg systemowy, który stanowi swego rodzaju unifikację elementów wywodzących się z różnych współczesnych wariantów UNIX-a (w *Książce* wymienia się m.in. Version 7, System III, System V i 4.3BSD). Aby umożliwić przenośność istniejących aplikacji świadomie zrezygnowano z tworzenia zupełnie nowego sprzęgu, choć są w standardzie oryginalne wynalazki jego autorów (np. nowa flaga *O_NONBLOCK*, wygodniejsza niż *O_NDELAY*).

● **Przenośność źródeł, nie binariów.** W standardzie w ogóle nie ma mowy o formacie binariów, czy to pośrednich, czy wynikowych.

● **Sprzęg minimalny.** Standard unika dublowania funkcji, chyba że w grę wchodzi ugruntowana tradycja (np. jest *creat*( ), szczególny przypadek *open*( ), ale nie ma *seekdir*( ) ani *telldir*( )). Nie ma w standardzie funkcji dotyczących administrowania systemem ani funkcji wykorzystywanych tylko przez superużytkownika. Tak więc każda rzeczywista implementacja standardu będzie musiała zawierać również funkcje spoza standardu

● **Sprzęg, nie implementacja.** W standardzie nie ma mowy o jakichkolwiek szczegółach implementacyjnych. Wszystkie stałe (numery sygnałów czy błędów) mają nazwy symboliczne, i tylko tak występują w standardzie. Nie precyzuje się metody realizacji funkcji systemowych (ekstrakody czy funkcje biblioteczne). Nic nie stoi na przeszkodzie, by funkcje standardu realizować drogą emulacji.

Szczegóły

Brzydka Zielona Książka składa się z dwóch części. Pierwsza z nich to sam standard (tylko 141 stron) i dodatki, druga jest obszernym komentarzem merytoryczno-historycznym. Co dokładniej zawiera standard? Przede wszystkim **definicje prawie stu funkcji systemowych**, w siedmiu grupach tematycznych (na marginesie: układ tematyczny to bodaj jedyne ustępstwo autorów standardu na rzecz przejrzystości):

● **Zarządzanie procesami**: *fork*( ), *execl*( ), *execv*( ), *execle*( ), *execve*( ), *execlp*( ), *execvp*( ), *wait*( ), *waitpid*( ), *_exit*( ), *kill*( ), *sigemptyset*( ), *sigfillset*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigaction*( ), *sigprocmask*( ), *sigpending*( ), *sigsuspend*( ), *alarm*( ), *pause*( ), *sleep*( ).
Funkcje z tej grupy umożliwiają tworzenie i elementarną synchronizację procesów, wykonywanie programów i korzystanie z sygnałów. Mechanizm sygnałów (tzw. niezawodne sygnały, ang. *reliable signals*) wzorowano głównie na rozwiązaniach z 4.3BSD, rezygnując jednak np. ze stosów sygnałów.

● **Środowisko procesów**: *getpid*( ), *getppid*( ), *getuid*( ), *geteuid*( ), *getgid*( ), *getegid*( ), *setuid*( ), *setgid*( ), *getgroups*( ), *getlogin*( ), *cuserid*( ), *getpgrp*( ), *setsid*( ), *setpgid*( ), *uname*( ), *time*( ), *times*( ), *getenv*( ), *ctermid*( ), *ttyname*( ), *isatty*( ), *sysconf*( ).
Ta grupa funkcji umożliwia procesom m.in. zasięganie informacji o sobie samych (identyfikatory procesu i ojca, użytkownika i grupy, itp) i o środowisku, w którym się wykonują (czas, zmienne środowiskowe, wartości systemowych zmiennych konfiguracyjnych, itp.).

● **Pliki i katalogi**: *opendir*( ), *readdir*( ), *rewinddir*( ), *closedir*( ), *chdir*( ), *getcwd*( ), *open*( ), *creat*( ), *umask*( ), *link*( ), *mkdir*( ), *mkfifo*( ), *unlink*( ), *rmdir*( ), *rename*( ), *stat*( ), *fstat*( ), *access*( ), *chmod*( ), *chown*( ), *utime*( ), *pathconf*( ), *fpathconf*( ).
Funkcje z tej grupy realizują usługi związane z tworzeniem i usuwaniem plików, katalogów i plików FIFO, umożliwiają informowanie się o ich cechach oraz zmienianie niektórych parametrów.

● **Wejście – wyjście**: *pipe*( ), *dup*( ), *dup2*( ), *close*( ), *read*( ), *write*( ), *fcntl*( ), *lseek*( ).
Są to funkcje realizujące operacje wejścia–wyjścia dotyczące plików i potoków oraz pewne operacje organizacyjne z tym związane. Tu nowością jest wspomniana już flaga *O_NONBLOCK*, podobna w działaniu do *O_NDELAY*, lecz umożliwiająca łatwe wykrycie zakończenia operacji z powodu końca pliku.

● **Sterowanie komunikacją**: *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( ), *tcsetattr*( ), *tcsendbreak*( ), *tcdrain*( ), *tcflush*( ), *tcflow*( ), *tcgetpgrp*( ), *tcsetpgrp*( ).
Funkcje z tej grupy realizują rozmaite operacje sterujące komunikacją asynchroniczną. Generalnie zastępują one tradycyjny mechanizm realizowany jako pojedyncza funkcja systemowa *ioctl*( ), i korzystają z nowej struktury systemowej *termios*, podobnej do *termio*, lecz bardziej uniwersalnej.

● **Powiązania z C**: *fileno*( ), *fdopen*( ), *sigsetjmp*( ), *siglongjmp*( ), *tzset*( ).
Funkcje z tej grupy wraz z rozszerzeniami do niektórych funkcji bibliotecznych ANSI C (np. *setlocale*( ), funkcje czasowe) umożliwiają programom pisanym w standardowym C gładką współpracę z POSIX-owym środowiskiem systemowym.

● **Systemowe bazy danych**: *getgrgid*( ), *getgrnam*( ), *getpwuid*( ), *getpwnam*( ).
Powyższe funkcje zapewniają dostęp do dwóch systemowych baz danych mieszczących informacje o poszczególnych użytkownikach i o grupach użytkowników.

Standard żąda, by prototypy funkcji systemowych (co najmniej te niezbędne) mieściły się w następujących **plikach nagłówkowych**:

<dirent.h>	<fcntl.h>	<grp.h>	<pwd.h>
<setjmp.h>	<signal.h>	<stdio.h>	<stdlib.h>

[1] The Institute of Electrical and Electronics Engineers, Inc., 1990 (Second Printing), ISBN 1-55937-003-3, 317 stron.

<sys/stat.h>	<sys/times.h>	<sys/utsname.h>	<sys/wait.h>
<termios.h>	<time.h>	<unistd.h>	<utime.h>

Zgodnie z UNIX-ową tradycją w przypadku **błędu** funkcje systemowe zwracają wartość – 1 lub NULL oraz nadają zmiennej *errno* wartość odpowiednią do sytuacji. Standard używa wyłącznie nazw symbolicznych dla wartości nadawanych zmiennej *errno* (przyporządkowanie wartości nazwom realizuje plik nagłówkowy <**errno.h**>). Lista możliwych błędów jest następująca:

[E2BIG],	[EACCES],	[EAGAIN],	[EBADF],	[EBUSY],
[ECHILD],	[EDEADLK],	[EDOM],	[EEXIST],	[EFAULT],
[EFBIG],	[EINTR],	[EINVAL],	[EIO],	[EISDIR],
[EMFILE],	[EMLINK],	[ENAMETOOLONG],	[ENFILE],	[ENODEV],
[ENOENT],	[ENOEXEC],	[ENOLOCK],	[ENOMEM],	[ENOSPC],
[ENOSYS],	[ENOTDIR],	[ENOTEMPTY],	[ENOTTY],	[ENXIO],
[EPERM],	[EPIPE],	[ERANGE],	[EROFS],	[ESPIPE],
[ESRCH],	[EXDEV].			

Standard definiuje również listę **sygnałów**, zarówno tych niezbędnych w każdej realizacji standardu:

SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2,

jak i tych, które każda implementacja musi definiować, lecz niekoniecznie realizować (jeśli jednak je realizuje, to w zdefiniowany standardem sposób):

SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU.

Wartości nazwom symbolicznym sygnałów nadaje plik nagłówkowy <**signal.h**>.

Ponadto każda implementacja może definiować **zmienne środowiskowe**, z których następujące – o ile zdefiniowane – muszą mieć ustalone w standardzie znaczenie:

HOME LANG LC_COLLATE LC_CTYPE LC_MONETARY LC_NUMERIC LC_TIME LOGNAME PATH TERM TZ

Implementacje standardu mogą istotnie różnić się między sobą wartościami nadawanymi rozmaitym **parametrom konfiguracyjnym**. Standard definiuje (w pliku nagłówkowym <**limits.h**> następującą listę wartości **minimalnych** dla rozmaitych parametrów konfiguracyjnych:

[POSIX_ARG_MAX]	4096
[POSIX_CHILD_MAX]	6
[POSIX_LINK_MAX]	8
[POSIX_MAX_CANON]	255
[POSIX_MAX_INPUT]	255
[POSIX_NAME_MAX]	14
[POSIX_NGROUPS_MAX]	0
[POSIX_OPEN_MAX]	16
[POSIX_PATH_MAX]	255
[POSIX_PIPE_BUF]	512

oraz wylicza podstawowe parametry konfiguracyjne wraz z wymaganiami co do ich wartości, formułowanymi w kategoriach powyższych wartości minimalnych (oczywiście implementacja może być „lepsza" niż wymaga tego standard).

Poza definicją sprzęgu systemowego w *Brzydkiej Zielonej Książce* znajdujemy również specyfikacje dwóch **formatów plików archiwalnych:** *tar* i *cpio* (tak więc stary konflikt zwolenników *tar* i *cpio* rozstrzygnięto w najprostszy sposób: postanowiono nie rozstrzygać). Oba formaty są rozszerzone w stosunku do swych tradycyjnych UNIX-owych pierwowzorów (szczegóły w pliku nagłówkowym <**tar.h**>). Najnowsze wersje większości UNIX-ów zawierają zwykle uniwersalne polecenia łączące funkcje klasycznych programów *tar* i *cpio* i pozwalające korzystać tak ze starych, jak i z nowych formatów archiwów (por. program ***pax*** w SunOS 4.1 [1]).

Zgodność ze standardem

Pojęcie zgodności ze standardem takim, jak standard sprzęgu systemowego ma dwa przystające do siebie znaczenia: czym innym jest zgodność „od dołu" – implementacji systemu, czym innym zgodność „od góry" – programu aplikacyjnego. W pierwszym przypadku sprawa jest prosta: system zgodny z *Brzydką Zieloną Książką*, czy krócej – ***zgodny z POSIX-em***, to system, który udostępnia aplikacjom sprzęg zawierający w sobie **między innymi** wszystkie funkcje i mechanizmy sprzęgu standardowego (plus wiele innych, niezbędnych dla normalnej eksploatacji systemu). Za przykład systemu w ten sposób zgodnego z naszym standardem może służyć **SunOS Release 4.1**, którego dokumentacja zawiera odrębny rozdział poświęcony *zgodności z POSIX-em* (por. [2]), precyzujący wszystko, co w standardzie pozostawiono do decyzji implementatorów (w tym również rozszerzenia w stosunku do standardu). Natomiast zgodność aplikacji ze standardem daje się stopniować w zależności od tego, czy, w jakim stopniu, i z jakich usług spoza standardu dana aplikacja korzysta (zainteresowanych bliżej tą kwestią odsyłam do *Książki)*.

Co dalej?

Standardy żyją swoim życiem: niektóre z czasem zmieniają się nie do poznania. *Brzydkiej Zielonej Książce* zmiany merytoryczne chwilowo nie grożą, raczej zanosi się na serię dodatków (a w nich m.in. powiązania symboliczne, obsługa komunikatów i być może jeszcze jeden format archiwów) i zmianę koloru okładki. Brzydsza chyba nie będzie.

LITERATURA

[1] SunOS Release 4.1 Reference Manual. Revision A of 27 March, 1990. Sun Microsystems, Inc., s. 401–404

[2] SunOS Release 4.1 System Services Overview. Revision A of 27 March, 1990. Sun Microsystems, Inc., s. 123–142.

JACEK SURMA

Programowanie obiektowe

dokończenie ze s. 12

Oczywiście, w ogólnym przypadku nie należy spodziewać się wartości równych lub bardzo bliskich *n*. Dociekliwy czytelnik mógłby również – i nie bez racji – zauważyć, że w pewnych ekstremalnych warunkach czas T_w może być większy od T_j. Ten aspekt sprawy nie będzie tu jednak omawiany. Niesłychanie istotną rolę dla czasu wykonania odgrywają algorytmy przydziału procesorów do poszczególnych obiektów. Wiele zależy też od rodzaju dysponowanego systemu wieloprocesorowego. Sytuacja komplikuje się wtedy, gdy nie wszystkie występujące w systemie procesory są jednakowe. Jeszcze więcej problemów przysparza sytuacja, gdy mamy do czynienia z systemem rozproszonym, na przykład siecią lokalną, gdzie większą rolę zaczynają odgrywać czasy transmisji. Strategie postępowania w wymienionych (i wielu innych) przypadkach są nieustannie rozwijane i udoskonalane.

Podsumowując, języki programowania zorientowane obiektowo nie są jedynym narzędziem, jakie można stosować przy tworzeniu aplikacji dla systemów wieloprocesorowych. Jednakże jest ono szczególnie przydatne do tego celu z uwagi na wiele swoich właściwości. Wystarczy w tym miejscu zauważyć, że wiele mechanizmów stosowanych w innych językach dla zapewnienia synchronizacji i wzajemnego wykluczania, jest zawartych w sposób niejawny w samej idei języków obiektowych. Czytelnikom zainteresowanym tematem, zaleciłbym lekturę marcowego zeszytu miesięcznika BYTE z 1989 r. (s. 228 i dalsze).

Jak wpływać na proces (w shellu Bourne'a)

Chodzi tu oczywiście nie o proces sądowy, lecz o proces UNIX-owy (w innych systemach nazywany także *zadaniem*). Procesy powołuje się w UNIX-ie niezwykle łatwo. Wydanie chociażby polecenia **ls**, wywołuje program **/bin/ls** jako oddzielny proces. Kod rozkazowy i dane wskazanego programu są zasadniczą częścią powołanego procesu, ale na cały proces składa się jeszcze szereg innych atrybutów, z których najważniejsze to:

- argumenty wywołania;
- deskryptory otwartych plików;
- obsługa sygnałów;
- bieżący skorowidz roboczy;
- identyfikator procesu (PID);
- identyfikator ojca procesu (PPID);
- identyfikator użytkownika rzeczywistego (UID);
- identyfikator grupy rzeczywistej (GID);
- identyfikator użytkownika efektywnego (EUID);
- identyfikator grupy efektywnej (EGID);
- wartość nice;
- maska uprawnień;
- limit długości plików;
- identyfikator grupy procesu;
- identyfikator grupy terminalowej (TTY);
- łączny czas pracy użytkownika i systemu.

Jak widzimy, gdy kod rozkazowy i dane staną się już częścią procesu, są one otulone w grubą kołdrę różnorakich atrybutów, zajmujących – to zależy od implementacji systemu – od kilkuset do kilkunastu tysięcy bajtów. UNIX pozwala nam skutecznie sterować tymi atrybutami i może to robić nawet nowicjusz UNIX-owy, który niedaleko wyszedł poza ogólną znajomość shella. Można modyfikować pewne atrybuty samego shella i są one później dziedziczone przez powoływane procesy. Inne atrybuty możemy specyfikować bezpośrednio dla powoływanych procesów. Na niektóre nie mamy żadnego wpływu i możemy tylko wypisywać sobie ich wartości. Popatrzmy jak to wygląda dla poszczególnych atrybutów.

Argumenty wywołania

Ten atrybut jest chyba najbardziej widoczny i oczywisty dla użytkownika: wołając jakikolwiek program opatrujemy go zawsze jakąś dłuższą lub krótszą listą argumentów (w szczególności pustą), np.:

```
cp /usr/dept locdept
ls
```

Argumenty pozycyjne muszą być zawsze wyspecyfikowane jawnie. Nieco więcej kłopotu jest z argumentami kluczowymi (*środowiskiem*). Mogą być one określane i jawnie i domyślnie, a zasady ich zestawiania przez shell są dość skomplikowane. Pisałem o tym dokładniej w 11–12 numerze *„Informatyki"* zeszłego roku.

Argumenty zajmują stosunkowo dużo miejsca w otulinie procesu: mogą to być tysiące bajtów.

Deskryptory otwartych plików

Shell ma stale otwarte trzy deskryptory plikowe:
0 – deskryptor standardowego wejścia;
1 – deskryptor standardowego wyjścia;
2 – deskryptor standardowych komunikatów.

W trybie interakcyjnym są one przypisane do terminala. Takie też deskryptory odziedziczy każdy wywołany przez nas proces, chyba że skierujemy wejścia–wyjścia procesu gdzie indziej, np.:

```
sort <dane_we >dane_wy 2>kom
```

W podobny sposób można utworzyć deskryptory o wyższych numerach, ale – w shellu Bourne'a – nie większych niż 9.

Obsługa sygnałów

Sygnał UNIX-owy to pewne uogólnienie przerwania sprzętowego. Sygnał może mieć obsługę domyślną (jest to po prostu zabicie procesu, który taki sygnał otrzymał), sygnał może być ignorowany i wreszcie może być przechwytywany z wykonaniem pewnej czynności. W shellu obsługę sygnałów ustawiamy poleceniem **trap**, np.:

```
trap 'rm /tmp/sh*; exit' 1 2 3
trap '' 2
trap 1 3
trap
```

Pierwsze z tych poleceń powoduje przechwytywanie wskazanych sygnałów 1, 2, 3 i wykonanie wskazanych instrukcji. Drugie spowoduje ignorowanie sygnału 2. Trzecie przywróci domyślną obsługę sygnałów 1 i 3. Ostatnie polecenie (bez argumentów) wypisze bieżącą obsługę sygnałów.

Kolejny proces odziedziczy obsługę danego sygnału, o ile sygnał jest obsługiwany domyślnie lub ignorowany. W innych przypadkach ustawiana jest obsługa domyślna.

Bieżący skorowidz roboczy

Skorowidz bieżący ustawiamy poleceniem **cd** np.

```
cd ../mycmd
```

i każdy powołany przez shella proces dziedziczy ostatnie ustawienie.

Identyfikator procesu (PID)

Nie mamy wpływu na ten atrybut. Identyfikator procesu bieżącego shella znajduje się w zmiennej $$. Możemy go wypisać np. poleceniem:

```
echo $$
```

Jeżeli wywołujemy jakiś proces w tle, np.:

```
diffeq data23 &
```

shell wypisze nam identyfikator tego procesu. Identyfikatory procesów i inne informacje o bieżących procesach uzyskamy za pomocą polecenia **ps**:

```
$ ps
PID TTY  TIME COMMAND
 32  01  0:04 sh
 41  01  0:01 ps
```

Identyfikator ojca procesu (PPID)

Na ten atrybut również nie mamy wpływu. Identyfikatorem ojca procesów powoływanych przez shell będzie oczywiście identyfikator

procesu shella. Identyfikatory ojców bieżących procesów uzyskamy za pomocą polecenia **ps -f**.

```
$ ps -f
 UID PID PPID  C STIME TTY TIME COMMAND
jasiek  32    1  0 10:51:40  01  0:04 -sh
jasiek  39   32 31 11:23:39  01  0:01 ps -f
```

Tylko w szczególny sposób możemy zmienić ten identyfikator. Mianowicie, jeżeli uda nam się zabić proces rodzicielski, osierocony proces zostanie zaadoptowany przez proces o numerze 1.

Uwaga: opcje polecenia **ps**, jak również format dostarczanych informacji, mogą się silnie różnić w poszczególnych wersjach UNIX-a.

Identyfikatory użytkownika i grupy (UID, GID, EUID, EGID)

Nie mamy wpływu na te atrybuty w odniesieniu do bieżącego shella. Możemy zmienić grupę za pomocą polecenia **newgrp**, ale związane jest to z powtórnym otwarciem sesji i będzie to już inne wcielenie shella.

Możemy wypisać te identyfikatory dla bieżącego shella za pomocą polecenia **id**:

```
$ id
uid = 50(jasiek) gid = 28(team2)
```

Jeżeli identyfikatory rzeczywiste i efektywne różnią się, polecenie wypisuje jedne i drugie.

Procesy dziedziczą identyfikatory rzeczywiste, natomiast ustanawianie identyfikatorów efektywnych jest bardziej skomplikowane i zależy od specjalnego uprawnienia **set-UID**.

Wartość nice

Wartość **nice** służy do wyliczania priorytetu wykonania według dość skomplikowanego algorytmu. Im wyższa wartość **nice**, tym niższy priorytet. Nie możemy zmienić tej wartości dla bieżącego shella, możemy natomiast zażądać wykonania programu z podwyższoną wartością **nice** wydając polecenie **nice**, np.:

```
nice -10 sort <dane_we >dane_wy
```

(*nice* – miły: używając tego polecenia jesteś „miły" dla innych).

Tylko super-użytkownik może obniżyć tę wartość.
Polecenie **ps -l** wypisuje wartość **nice** dla bieżących procesów.

Maska uprawnień

Maska uprawnień określa atrybuty kreowanych plików. Maskę ustawiamy dla bieżącego shella poleceniem **umask**. Argumentem polecenia jest trzycyfrowa liczba ósemkowa: kolejne cyfry tej liczby odnoszą się do uprawnień: *właściciela, grupy* i *innych*. Bity każdej z tych cyfr określają odpowiednio uprawnienia do *czytania/pisania/wykonania*. Jedynka oznacza odebranie odpowiednich uprawnień. Na przykład polecenie:

```
umask 177
```

odbiera właścicielowi uprawnienia do wykonywania dalej tworzonych plików, a grupie i innym – odbiera wszelkie uprawnienia do tych plików. Powoływane później procesy dziedziczą tak ustawioną maskę.

Polecenie **umask** bez argumentów wypisuje bieżącą maskę.

Limit długości plików

Limit długości plików możemy ustawić dla bieżącego shella poleceniem **ulimit**, np.:

```
ulimit 100
```

Argument określa maksymalną liczbę bloków dla zapisywanych plików. Zależnie od systemu, rozmiar bloku może wynosić 512B, 1024B lub więcej. Powołany w shellu proces dziedziczy tak ustawiony limit.

Polecenie **ulimit** bez argumentów wypisuje bieżący limit.

Atrybut ten nie wpływa na rozmiar czytanych plików: wolno czytać pliki o nieograniczonej z góry długości.

Identyfikator grupy procesów

W zbiorze wszystkich bieżących procesów wyróżniamy grupy procesów: pomijając super-użytkownika, procesy mogą sobie przesyłać sygnały tylko wewnątrz grupy. W dynamicznie powstającej hierarchii procesów można powoływać nowe grupy procesów, jednakże nie jest to możliwe na poziomie shella.

Identyfikator grupy terminalowej (TTY)

Ten atrybut określa, który terminal jest terminalem sterującym danego procesu. Sygnały pochodzące z jakiegoś terminala dochodzą tylko do tych procesów, dla których jest on terminalem sterującym. W shellu nie mamy wpływu na ten atrybut – możemy go tylko wypisać poleceniem **ps**.

Wszystkie procesy grupy terminalowej tworzą również tę samą grupę procesów, dopóki któryś z nich nie utworzy oddzielnej grupy procesów.

Łączny czas pracy użytkownika i systemu

Czasy te dla dotychczasowych poleceń bieżącego shella możemy wypisać za pomocą polecenia **times**. Możemy zażądać wykonania konkretnego polecenia z wyliczeniem tych czasów za pomocą polecenia **time**, np.:

```
time sort <dane_we >dane_wy
```

JAN WALASEK

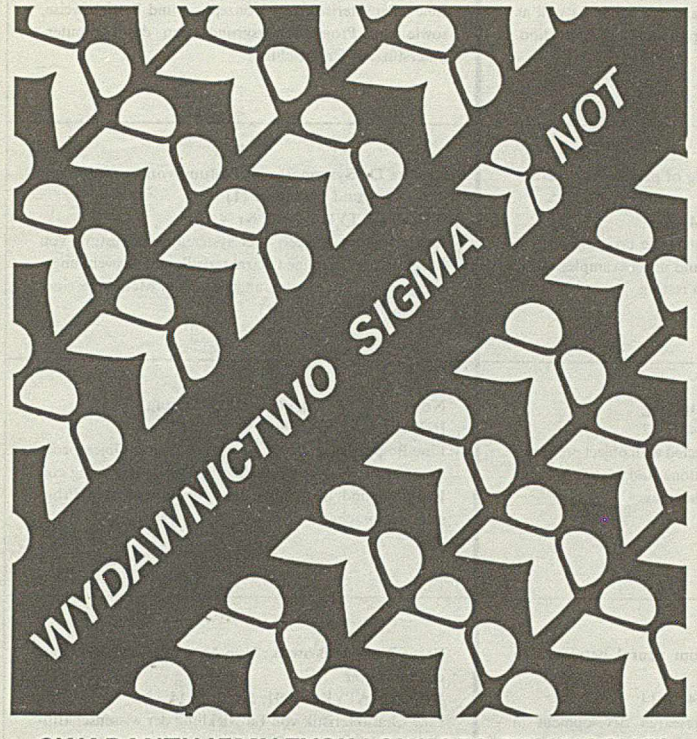
Komputery równoległe

Firma Intel zastąpiła w swej rodzinie równoległych superkomputerów typu RISC, znanych pod nazwą HyperCube, stosowane dotąd procesory 80386 nowymi procesorami o symbolu 80860. Występują one w systemach komputerowych zawierających od 8 do 128 procesorów. Cena tych systemów wynosi od 265 tys. do jednego mln dolarów, a ich moc obliczeniowa może dochodzić do 7,6 miliarda operacji na sekundę.

Jednocześnie firma Masper Computer Corp. z Sunnyvale w Kalifornii ogłosiła dane dotyczące rodziny systemów o symbolu MP-1, która obejmuje osiem modeli umożliwiających implementację do 16 384 procesorów. Systemy takie są budowane z zastosowaniem kostek opracowanych na indywidualne zamówienie i zapewniają uzyskanie 30 mld operacji na sekundę (lub 1,5 mld operacji zmiennoprzecinkowych (s). Ceny wynoszą od 170 tys. dolarów za 1024-procesorowy system MP 1101 do 810 tys. dolarów za 16 384-procesorowy system MP 1216. (J.R)

REKLAMA U NAS

zapewnia możliwość dotarcia do zainteresowanego i wyspecjalizowanego odbiorcy



GWARANTUJEMY ZYSK I SUKCES FINANSOWY

fachową i sprawną obsługę, doradztwo,
promocję wyrobów i usług na łamach
61 czasopism technicznych

NIE CZEKAJ! TELEFONUJ: 20-31-24

Wybierz jedno z naszych czasopism!
Ceny konkurencyjne!

1. AUDIO-HIFI-VIDEO
2. AURA
3. CEMENT, WAPNO, GIPS
4. CHEMIK
5. CHŁODNICTWO
6. CIEPŁOWNICTWO, OGRZEWNICTWO, WENTYLACJA
7. DOŚWIADCZONY MISTRZ
8. DOZÓR TECHNICZNY
9. ELEKTRONIZACJA
10. ELEKTRONIKA
11. ENERGETYKA
12. GAZ, WODA I TECHNIKA SANITARNA
13. GAZETA CUKROWNICZA
14. GOSPODARKA MIĘSNA
15. GOSPODARKA PALIWAMI I ENERGIĄ
16. GOSPODARKA WODNA
17. HUTNIK
18. INFORMATYKA
19. INŻYNIERIA I BUDOWNICTWO
20. INŻYNIERIA MATERIAŁOWA
21. INŻYNIERIA MORSKA
22. KOKS, SMOŁA, GAZ
23. MATERIAŁY BUDOWLANE
24. MÓJ DOM
25. MODA
26. NOWATOR
27. OCHRONA POWIETRZA
28. OCHRONA PRACY
29. OCHRONA PRZED KOROZJĄ
30. ODZIEŻ
31. OPAKOWANIE
32. PROBLEMY JAKOŚCI
33. PRZEGLĄD ELEKTROTECHNICZNY
34. PRZEGLĄD GASTRONOMICZNY
35. PRZEGLĄD GEODEZYJNY
36. PRZEGLĄD GÓRNICZY
37. PRZEGLĄD ODLEWNICTWA
38. PRZEGLĄD PAPIERNICZY
39. PRZEGLĄD PIEKARSKI I CUKIERNICZY
40. PRZEGLĄD POLSKIEJ TECHNIKI
41. PRZEGLĄD SKÓRZANY
42. PRZEGLĄD TECHNICZNY
43. PRZEGLĄD TELEKOMUNIKACYJNY
44. PRZEGLĄD WŁÓKIENNICZY
45. PRZEGLĄD ZBOŻOWO-MŁYNARSKI
46. PRZEMYSŁ CHEMICZNY
47. PRZEMYSŁ DRZEWNY
48. PRZEMYSŁ FERMENTACYJNY I OWOCOWO-WARZYWNY
49. PRZEMYSŁ SPOŻYWCZY
50. RADIOELEKTRONIK
51. RUDY I METALE NIEŻELAZNE
52. SZKŁO I CERAMIKA
53. TECHNIK WŁÓKIENNICZY
54. TECHNIKA ZAGRANICZNA
55. WIADOMOŚCI ELEKTROTECHNICZNE
56. WIADOMOŚCI GÓRNICZE
57. WIADOMOŚCI HUTNICZE
58. WIADOMOŚCI TELEKOMUNIKACYJNE
59. WIADOMOŚCI PRODUKCYJNE
60. HORYZONTY TECHNIKI ZRÓB SAM
61. WIADOMOŚCI I PROPOZYCJE

WYDAWNICTWO SIGMA-NOT Sp. z o.o.
DZIAŁ REKLAMY I MARKETINGU
ul. Biała 4, 00-950 Warszawa
tel. 20-31-24 tlx 814550 fax 20-31-16

MODEMY

- homologowane, 2400/1200/300 b/s, na łącza komutowane i dzierżawione, wolnostojące i na karcie do PC, także zgodne ze standardem Hayes

oraz

- konwertery 9600 b/s baseband
- konwertery RS232 – pętla prądowa
- tester V24/RS232 – pocket size
- programator EPROM 2716-27512 oraz mikroprocesorów jednoukładowych rodziny MCS48, MCS51
- interfejsy pomiarowe IEC 625 (IEEE 488) do PC
- pakiet transmisji synchronicznej BSC do PC
- oprogramowanie emulacyjne stacji IBM 3780
- koncentrator stanowisk operatorskich do MERY 9150
- adapter telekomunikacyjny do ODRY 1305

oferuje

SWEDEX UNIVERSAL Co. LTD
ul. Wspólna 1
05-075 Wesoła k. Warszawy
tel. (0-22) 153-365
teleks 813935 swedx pl

ZO/236/90-A

PROMOCYJNA
80% OBNIŻKA CEN!

LOGOTEC® Engineering

Oferujemy

licencjonowane, polskie wersje językowe
zachodnioniemieckich systemów wspomagania projektowania

LOGOCAD – mechanika **ELOCAD** – elektrotechnika

oprogramowanie pomocnicze

pakiety ponad 100 polskich (PN) i niemieckich (DIN)
norm mechanicznych (śruby, nakrętki, podkładki, pierścienie,
zawleczki, sworznie, sprężyny, łożyska toczne itp.)

inżynierskie programy obliczeniowe:

(wały, belki, ramy, kraty, ruszty, płyty, powłoki,
płaski stan naprężeń i płaski stan odkształceń,
przekroje cienko- i pełnościenne, obliczenia i optymalizacja kół zębatych)

programy do obsługi baz danych

UNIP2 – uniwersalny generator wydawnictw
i programów standardu dBase i Clipper
nagrodzony MikroLaurem '88 i '90
ARCHIW – archiwum rysunków i projektów
MOL(książkowy) – do kompleksowej obsługi biblioteki
i wypożyczalni książkowej

doradztwo i szkolenia

z zakresu zastosowania naszych systemów CAD,
kursy obsługi naszego oprogramowania,
pomoc we wdrożeniu i eksploatacji programów

sprzęt

kompletne stanowiska dla naszych systemów,
digitizery, monitory i karty graficzne
dla różnych systemów CAD.

Jesteśmy do państwa dyspozycji

LOGOTEC ENGINEERING
Zakład Rozwoju Systemów CAD
Nasz nowy adres: 40-028 KATOWICE, ul. Francuska 70
tel.: 57-46-26, 57-23-76, 51-17-47, teleks 0312606

0/21/90

Piecha J.: Systemy CAI INFORMATYKA 1991, nr 4, s. 1 Charakterystyka koncepcji oraz zasady działania i projektowania systemów nauczania wspomaganego komputerem.	**Piecha J.: CAI systems** INFORMATYKA 1991, No. 4, p. 1 Characteristics of conception and operating, as well as designing principles of the computer assisted instruction.	**Piecha J.: CAI-Systems** INFORMATYKA 1991, Nr. 4, S. 1 Eine Charakteristik von Konzeption und Betriebsweise, sowie von Projektierungsgrundsätzen des computerunterstützten Unterrichts.
Kuchta D.: Systematyczne testowanie programów – zalety i wady (1) INFORMATYKA 1991, nr 4, s. 8 Pierwsza część artykułu o systematycznym testowaniu programów, zawierająca charakterystykę i przykłady użycia najczęściej stosowanych metod takiego testowania.	**Kuchta D.: Systematic testing of programs – advantages and disadvantages (1)** INFORMATYKA 1991, No. 4, p. 8 First part of the article on systematic program testing, which includes characteristics and using examples of the most applied methods for such testing.	**Kuchta D.: Systematische Testung von Programmen – die Vor- und Nachteile (1)** INFORMATYKA 1991, Nr. 4, S. 8 Erster Teil des Artikels über systematische Testung von Programmen, der eine Charakteristik und Anwendungsbeispiele der am meisten angewendeten Methoden einer solchen Testung umfasst.
Nowaczyk L.: Programowanie obiektowe INFORMATYKA 1991, nr 4, s. 12 Omówienie podstawowych pojęć związanych z programowaniem obiektowym z podkreśleniem funkcji i znaczenia obiektowo ukierunkowanych języków programowania.	**Nowaczyk L.: Object programming** INFORMATYKA 1991, No. 4, p. 12 Discussion of basic terms connected with object programming with emphasis on functions and significance of object oriented programming languages.	**Nowaczyk L.: Objektprogrammierung** INFORMATYKA 1991, Nr. 4, S. 12 Eine Besprechung der grundsätzlichen, mit Objektprogrammierung verbundenen Begriffen unter Betonung von Funktion und Bedeutung der objektorientierten Programmiersprachen.
Berus T., Macukow B.: Od sieci neuronowych do neurokomputerów INFORMATYKA 1991, nr 4, s. 14 Charakterystyka rozwoju badań naukowych w dziedzinie modelowania i konstruowania sieci neuronowych oraz omówienie koncepcji neurokomputera i jego cech w porównaniu do komputera klasycznego.	**Berus T., Macukow B.: From neural networks to neurocomputers** INFORMATYKA 1991, No. 4, p. 14 Characteristics of scientific research development in neural networks modelling and building, as well as discussion of neurocomputer concept and its features in comparison with classic computer.	**Berus T., Macukow B.: Von Neuronnetzen bis Neuroncomputer** INFORMATYKA 1991, Nr. 4, S. 14 Eine Charakteristik von Entwicklung der wissenschaftlichen Forschung im Bereich der Modellierung und Konstruktion von Neuronnetzen, sowie eine Besprechung der Konzeption von Neuroncomputer und seiner Eigenschaften in Vergleich mit einem klassischen Computer.

SIMPLE® Sp. z o.o.

to organizacja + informatyka to największy w Polsce Software House

Oferujemy wszystkie dostępne na rynkach zachodnich pakiety oprogramowania licencjonowanego po bardzo niskich cenach. I tak np.:

Microsoft
- Quick C wersja 2.0 – 1,5 mln zł
- Windows wersja 3.0 – 2,5 mln zł

Borland International
- Turbo PASCAL wersja 6.0 Professional – 3,9 mln zł
- Turbo C wersja 3.0 Professional – 3,9 mln zł

Peter Norton Computing
- Norton Commander wersja 3.0 – 2,1 mln zł
- Norton Utilities wersja 5.0 – 2,5 mln zł

Nantucket
- Clipper wersja 5.0 –7,5 mln zł

Pełny katalog oprogramowania (aktualizowany co miesiąc) oferujemy w cenie 50 tys. zł lub udostępniamy w siedzibie firmy. Realizujemy każde zamówienie w terminie do trzech tygodni.

Oferujemy poza tym własne oprogramowanie użytkowe przedsiębiorstw, przeznaczone do pracy w sieci firmy Novell, opracowane na podstawie oryginalnej bazy danych SBase, nowoczesne narzędzia – język C, pakiet dialogu Vermont Views. Zapewniamy szkolenie i wersje źródłowe oprogramowania.

Pamiętaj: Simple to dobry business!

SIMPLE Sp. z o.o. 04-541 Warszawa ul. Karpacka 14 c

Zakład Informatyki ul. Karpacka 12 tel.: 15-49-83 faks: 635-91-66

SO/84/91

Wszystkich zainteresowanych kupieniem INFORMATYKI prosimy o kontakt z redakcją (tel. 39-14-34) lub Działem Handlowym SIGMY (tel. 40-30-86) – możemy wysyłać egzemplarze za zaliczeniem pocztowym. Czytelnicy z terenu Warszawy mogą kupić poszczególne numery w Klubie SIGMY przy ul. Mazowieckiej 12, w księgarni PP „Domu Książki" przy ul. Mokotowskiej 51/53 oraz w lokalu redakcji.

Jednocześnie przypominamy, że najpewniejszą formą otrzymywania INFORMATYKI jest prenumerata.

KARTY PRZETWORNIKÓW ANALOGOWO-CYFROWYCH
do komputerów IBM XT/AT/386
Podzespoły ANALOG DEVICES MAXIM BURR BROWN

- 12, 14, 22 bity
- izolacja galwaniczna
- pamięć buforowa
- praca „w tle"

próbkowanie do 1 MHz!!!

KARTY PROCESORÓW SYGNAŁOWYCH
TMS320C25 MOTOROLA 56001 AT&T DSP32C

Bogate oprogramowanie
Kompleksowa obsługa zamówień
Serwis ★ Gwarancja ★ Pomoc w instalacji

PRACOWNIA USŁUG INFORMATYCZNYCH I KOMPUTEROWYCH 'CONVERT'
50-412 Wrocław, ul. Mazowiecka 17,
telefon 300-11 w. 210, 44-85-66 telefaks 44-85-66

O/4/91

COLOR Gra

Parę lat temu Ministerstwo Obrony Stanów Zjednoczonych ogłosiło konkurs na opracowanie nowego języka programowania. Składane projekty były opatrzone hasłami – kolorami czerwony, zielony, niebieski. Po dogłębnych analizach propozycji, w trakcie których odpadali kolejni pretendenci do wygranej – wybrano wreszcie jedno rozwiązanie. Po zdarciu kolorowego hasła, okazało się, że autorem nowego języka, nazwanego później językiem Ada, jest Pan Ichbiach z Francji. Amerykanie jakoś przełknęli ten, zdawałoby się, narodowy afront i postanowili potraktować nowy język z całym szacunkiem. Oczywiście, zanim wyschła farba na raporcie opisującym nowy standard militarny, pojawiło się wiele artykułów krytykujących wybrane rozwiązanie. Po uwzględnieniu najważniejszych poprawek, język Ada stał się oficjalnym językiem programowania dla armii amerykańskiej, która składa się z trzech wzajemnie zwalczających się sił (wojska lądowe, marynarka, lotnictwo).

Zostawmy jednak im to zmartwienie i przenieśmy się do innego kraju, który w ostatnim czasie szybko zmienił swoje barwy (i dalej nie może się jeszcze zdecydować, w jakim kolorze będzie mu do twarzy). W kraju tym rok temu ogłoszono konkurs na przemalowanie z całkowitą renowacją dużej powierzchni, surowej lub pokrytej starymi farbami w różnych kolorach i o wielu fakturach.

Do konkursu stanął niebieski, szary, czarny, czerwony i cała tęcza różnokolorowa. Zielony nie zauważył ogłoszenia, a więc się trochę spóźnił. Wszyscy proponowali według nich najlepsze rozwiązanie, ale tak naprawdę to nikt nie wiedział co należy zrobić, jak dużo miejsc trzeba będzie pomalować. Najpierw odpadli mało kolorowi – mieli za mało farby. Potem szary nie chciał odpowiadać za jakość koloru po pomalowaniu, bał się, może nawet i słusznie, czy jest w stanie pokryć starą o nieznanej fakturze farbę, a nowa faktura była jeszcze (i jest dalej) w opracowywaniu.

Niebieski proponował farby w zamkniętej puszce, twierdząc, że dają one najlepszą jakość. Można się było z tym zgodzić, ale do wszelkich retuszy i podmalowanek trzeba by w przyszłości wzywać niebieskiego z jego zamkniętymi puszkami. Proszono go o puszki otwarte, ale nie chciał ich zaproponować, bo może wolał upłynnić zapasy.

Pozostali proponowali puszki otwarte. Przynajmniej w przyszłości można będzie mieszać farby. Wybierającym trudno jednak było określić, jaki kolor uzyskają po wyschnięciu farby. Również zaproszeni w ostatniej chwili krytycy sztuki nie bardzo chcieli im pomóc, narzekając, że nie zostały określone warunki malowania. Zresztą kryteria doboru barw są przecież tak subiektywne. Nie każdy jest przecież artystą malarzem, a ci często mają ustalone, bardzo wąskie, poglądy. Niedawno widziałem w galerii obrazy tylko w tonacji krwistoczerwonej.

Wreszcie dokonano wstępnego wyboru. Wybierający byli tak już umazani, że niektórzy z nich nie chcieli się już dalej truć (jak bardzo farby mogą zaszkodzić, to wiedzą ci, którzy oglądają „Dynasty"). Najbardziej odporni na przeciągi wybrali zielonego. Co prawda ekolodzy twierdzą, że zielony jest bardzo słaby i ma kłopoty. Za zielonym stoi jednak cały las, który w każdej chwili może mu pomóc. Poza tym, jeżeli zajrzymy do puszek zielonego to możemy zobaczyć, że blacha, powłoka oraz esencja pochodzi z kontynentu niebieskiego. Na wszelki wypadek zielony został związany silniejszymi więzami kontraktu i dostał trochę czasu na pokazanie swoich umiejętności. Jeżeli próbka nie będzie nam odpowiadała to nie dostanie pieniędzy na następne pędzle i straci możliwość pomalowania całości.

Tymczasem niebieski, zrobiony na zielono, zaczął poszukiwać innych dróg odzyskania swojej pozycji. Krytyków znalazł na forum przekazując im solenne stwierdzenie, że potrafi najlepiej malowań oraz, że może nawet zaoferować nowe – już otwarte puszki (faktycznie te jego nowe puszki są całkiem niezłe, ale dlaczego pojawiły się tak późno?). A ponieważ forum akurat podzieliło się na różne odcienie tej samej barwy, to znalazł tam posłuch i zaczęto poszukiwać dziury w całym. Na forum wezwano wybierających i zaczęto im zadawać pytania dlaczego polubili zielonego i czy nie zanurzyli wcześniej rąk w zielonych. Niczego konkretnego jednak nie stwierdzono.

Jakie wnioski płyną z tej gry w kolory. Po pierwsze przed poszukiwaniem najlepszych farb należy dobrze określić stan powierzchni do pomalowania oraz sprecyzować dokładnie jaki kolor i fakturę chcemy uzyskać. Następnie trzeba rozesłać kwerendę do wszystkich potencjalnych malarzy, zapraszając ich do konkursu. Potem można zacząć weryfikować kolory, dbając aby nie zabrudzić sobie rąk. Warto wtedy poszukać doświadczonych ekspertów, żądając od nich sprecyzowania kryteriów oceny i opinii końcowej. Oczywiście, opinie takie będą zawsze subiektywne, wszak nikt nie jest w stanie obiektywnie ocenić kolorów. Tym niemniej mając kilka takich opinii oraz własny zestaw kryteriów, zweryfikowany kryteriami ekspertów, łatwiej jest dokonać wyboru. Niestety zawsze trzeba się liczyć z faktem, że znajdą się oponenci z innej szkoły oraz również daltoniści nie przyznający się, nawet przed sobą, do swojej ułomności.

Na razie gra w kolory toczy się dalej.

WACŁAW ISZKOWSKI

P.S. Wszelkie podobieństwo faktów i zjawisk opisanych w tym felietonie do rzeczywistych osób, instytucji i zjawisk jest całkowicie przypadkowe.
zBITki poglądów i spostrzeżeń, tutaj prezentowanych, nie są oficjalnym stanowiskiem Redakcji ani żadnej innej organizacji. Ich prezentacja ma służyć pobudzeniu dyskusji na dany temat nawet przez wzburzenie moich przyszłych adwersarzy (WI).

PRZEDSIĘBIORSTWO
ZASTOSOWAŃ INFORMATYKI
meditronik
oferuje

UM7106	18 000
UM3481,2,3,4,10,11	8200
UM66T	3600
UM82450	14 000
UM8250B	14 000
UM8253-5	16 500
UM8259A-2	27 300
UM82C11	13 000
UM82C55A	16 500
UM82C8167	19 500
UM6116U-35(ns)	25 500
UM6116-2L	11 500
UM6264-10	20 000
UM62256A-10L	52 000

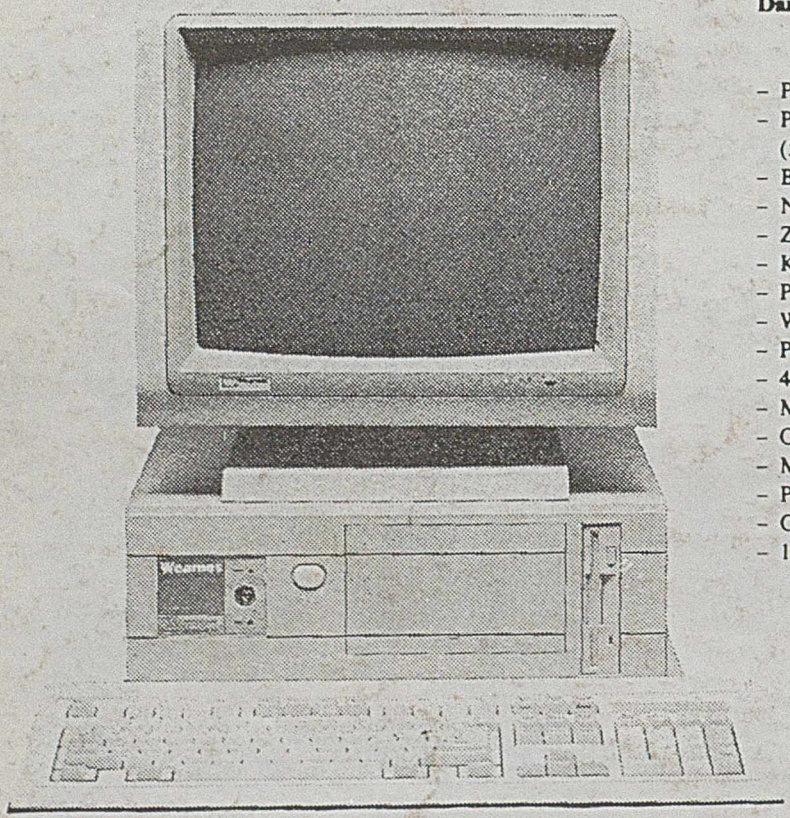
Nasz adres:
PZI „MEDITRONIK"
00-194 Warszawa, ul. Dzika 4
telefon 635-22-63, 635-22-64
telefaks 635-21-95
teleks 816075 medi pl
0/24/90

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
telefon i telefaks: 518-628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. M. Curie Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 35325

O/22/90